150 ptas.

39

# mi computer

## CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR




Editorial Delta, S.A.

# MI COMPUTER CURSO PRACTICO
## DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen IV - Fascículo 39



Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.


ISBN: 84-85822-83-8 (fascículo) 84-7598-005-8 (tomo 4)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 108410
Impreso en España - Printed in Spain - Octubre 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

**Servicio de suscripciones y atrasados** (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

**No se efectúan envíos contra reembolso.**

Editorial Delta, S.A.



# A la orden de sus dedos

**Las bases de datos constituyen un sotfware fundamental. Vamos a examinarlas en sus versiones para microordenadores**

Una *base de datos* es un conjunto de datos relativos a un tema. Por ejemplo, una base de datos podría mantener uno de los siguientes conjuntos de información: los nombres y las direcciones de los socios de un club; los gastos que representan sus reuniones; las fechas y lugares de tales reuniones, o el pago de las cuotas sociales efectuadas por los socios del club. Sin embargo, un sistema de base de datos podría incluir todos estos datos en un gran archivo.

Un *archivo* es un conjunto de registros, cada uno de los cuales se compone de un cierto número de campos. En algunas aplicaciones, como en los paquetes de contabilidad, la estructura del archivo se determina por el software, pero en un sistema de base de datos es más común que la estructura del archivo se elija en función de la tarea que lo va a explotar. Esto implica determinar el tamaño de cada campo y definir su clase de contenido. En un archivo de nombres y direcciones, por ejemplo, los detalles relativos a cada persona ocupan un registro individual: su nombre se almacena en un *campo de tipo carácter* o *alfanumérico* y cada línea de su dirección en campos separados. Estos campos suelen tener treinta caracteres de longitud.

La alternativa a un campo alfanumérico es un *campo numérico*, que permite al programa efectuar cálculos con los datos almacenados en él. En nuestro archivo de socios del club, el programa podría calcular cuántos miembros han pagado sus cuotas, a cuánto asciende el total de ingresos en lo que va de año, o cuánto queda aún por cobrar. No obstante, no todos los datos numéricos han de ser almacenados en campos numéricos. Los números de teléfono constituyen un buen ejemplo de números con los que no se efectúa ningún cálculo, y éstos se almacenan normalmente en campos alfanuméricos.

Para saber si en su ordenador se puede utilizar cierto archivo, es necesario calcular las dimensiones máximas del mismo (cuántos registros deberá tener), calcular la cantidad de memoria necesaria para cada registro y luego determinar si se dispone de suficiente memoria para almacenar el archivo. Un registro con un nombre y tres campos de dirección de 30 caracteres cada uno y un número de teléfono que ocupe 10 caracteres ocupa un total de 130 bytes. Si se tiene una unidad de disco en el sistema con 200 Kbytes de capacidad, entonces podrá guardar 1 500 registros en cada disco. En un sistema sin unidades de disco con 48 Kbytes de memoria sólo habrá espacio para 300 registros (concediendo alre-

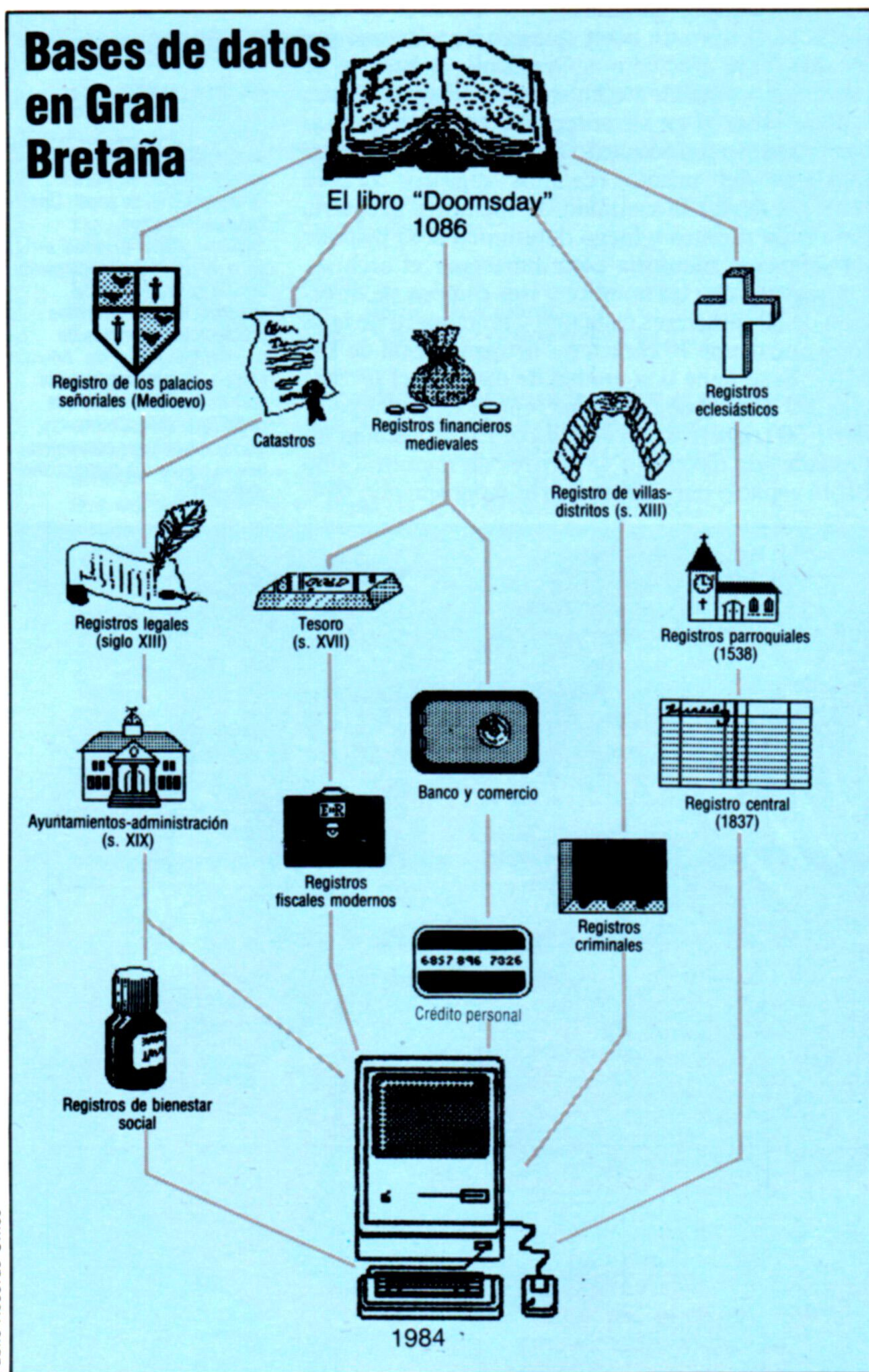
**Archivos caseros**
La mayoría de las personas conservan información acerca de sí mismas archivadas sin ningún método en trozos desperdigados de papel. Una base de datos por microordenador personal sería útil a modo de almacenamiento central de las pólizas de seguros, detalles sobre las propiedades, números de cuentas bancarias, etc. Además, tendría un valor incalculable para cualquier persona que coleccione sellos, monedas, discos o cualquier otro objeto que se preste a la clasificación sistemática

Mike Brownlow

dedor de 10 Kbytes para el programa y el sistema operativo). En la práctica, si se desea clasificar el archivo o hacer otro tipo de tratamiento de esta clase, será necesario algún espacio de trabajo libre, y es conveniente limitar el tamaño del archivo a la mitad de la zona disponible. Los dos sistemas de almacenamiento difieren de forma tan radical debido a que los archivos en cinta se deben leer en la memoria y procesar como un todo, mientras que la velocidad de los accesos a archivos en disco permite que los archivos residan en disco y se procesen en la memoria.

Un sistema de gestión de datos permitiría tomar información de un archivo (el total de gastos del año, p. ej.) y relacionarla o compararla con información de otro archivo, como por ejemplo, el total ingresado en concepto de cuotas sociales. Basándose en esa información, se podría, por ejemplo, tomar una decisión acerca de si iniciar o no una campaña de captación de socios, reducir la cantidad de reuniones o invertir los beneficios en un nuevo ordenador para el club. Quizá podría ser conveniente enviar una carta estándar a los socios del club informándoles de la situación. La base de datos podría proporcionar los nombres y las direcciones, permitiendo su impresión directamente en sobres o etiquetas autoadhesivas. O bien podría enlazarlas con un paquete para tratamiento de textos para escribir cartas e informes. A pesar de que son muchas las cosas que puede hacer la base de datos ideal, no resulta fácil encontrar una que ofrezca todas las facilidades juntas, especialmente si se posee un ordenador personal con un espacio de memoria limitado o el almacenamiento es en cinta y no en disco. Muchos de los paquetes de bases de datos menos sofisticados sólo tratan un archivo cada vez, de modo que aunque se pudieran retener la totalidad de los distintos grupos de información que hemos mencionado antes, no podrían ser relacionados ni comparados en un auténtico estilo de base de datos.

**Sólo basta conectar**
Las bases de datos proliferan en las sociedades técnicas que dependen de la acumulación de hechos, y la accesibilidad de la información crece a medida que la tecnología avanza. En el pasado, las bases de datos estaban aisladas unas de otras en virtud de la distancia y el tiempo, pero ahora que tantas bases de datos privadas y de la administración están basadas en ordenador, la posibilidad de que éstas se comuniquen entre sí supone una amenaza potencial para la intimidad y el derecho a la vida privada de las personas



Ian McKinnell

En todo proceso de datos existe el riesgo de depositar toda una valiosa información en el ordenador y que, a causa de algún accidente, se destruya. Cuando se trata de información importante es esencial contar con copias de seguridad.

Un paquete de base de datos sencillo tratará un único archivo de información. El sistema hará que el trabajo rutinario de introducir los datos en el archivo sea tan fácil como rellenar un formulario, y permitirá el acceso a la información tanto a través de la pantalla como de una impresora. El usuario debería ser capaz de seleccionar registros individuales a partir de criterios de búsqueda y clasificación, por ejemplo "todos los socios que todavía no han pagado este año y que tienen un código postal determinado". Un paquete de esta clase también deberá ser capaz de imprimir campos específicos de un registro, como los campos de nombre y dirección solamente, para hacer listas de correspondencia. Existen paquetes sencillos de este tipo para la mayoría de los ordenadores personales basados en cinta. Sin embargo, si se posee un sistema de disco, la gama del software disponible es mucho mayor.

Aparte de los usos que ya hemos sugerido, en los ordenadores personales los sistemas de base de datos se pueden utilizar para catalogar colecciones de discos, libros, referencias bibliográficas, colecciones de sellos, etc.

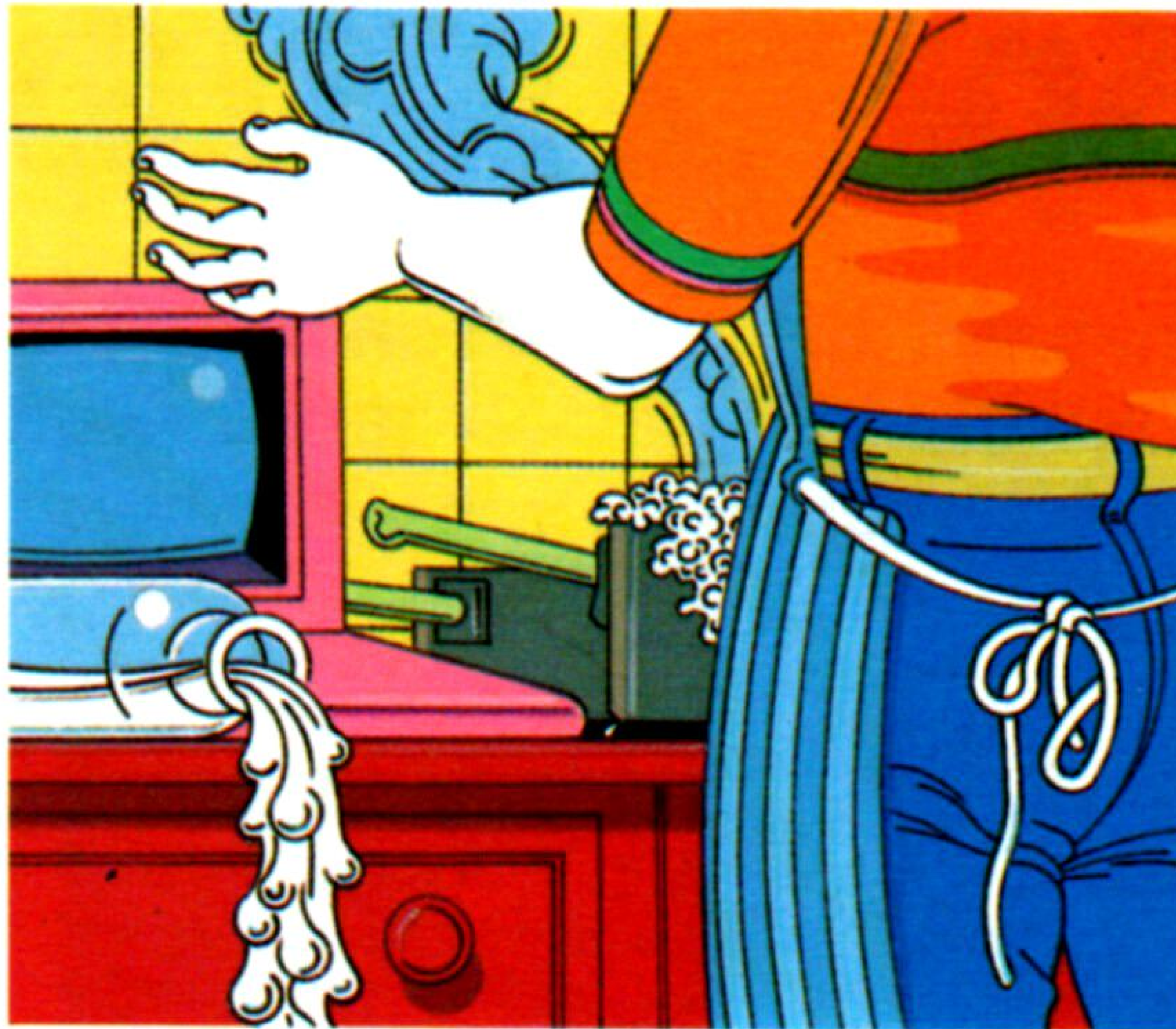
Los paquetes ligeramente más sofisticados ya permiten diseñar un formulario de entrada de datos por pantalla a tono con un sistema previo basado en papel. Otros paquetes permitirán seleccionar ítems numéricos que sean mayores o menores que una cantidad enunciada (como todos los socios que deben más de 2 000 pesetas, p. ej.). Por consiguiente, es importante rebuscar un poco por las tiendas hasta encontrar el paquete que se adapte mejor a las necesidades específicas.

Un ordenador con un paquete de base de datos es útil para tareas que implican repetición, o cuando hay que clasificar o buscar rápidamente mucha información. Pero hay otras aplicaciones que no son en absoluto adecuadas para los sistemas de base de datos. No parece sensato tener que buscar en un archivo de números de teléfono con un ordenador personal cada vez que uno desea hacer alguna llamada telefónica. En el tiempo que le lleva a usted conectar el ordenador, cargar el programa de



## Receta para el desastre

Juan tiene un Vic-20 con el que está muy encariñado. Le ha ayudado a entender los ordenadores y a aprender algo de programación en BASIC, pero lamentablemente Juan no puede distinguir entre lo que es una aplicación seria y lo que es entretenimiento. Insiste en utilizar el ordenador para guardar los números de teléfono de sus (pocos) amigos, y usa una base de datos para almacenar sus recetas de cocina. Cuando uno va a cenar a su casa, invariablemente no puede comer nada hasta pasada la medianoche, porque Juan insiste en verificar los detalles de las instrucciones culinarias en su Vic. Lo enchufa con gesto dramático, carga el programa desde la cinta (para conseguirlo suele realizar varios intentos), espera alegremente durante el interminable período en que el ordenador busca y lee, contempla las instrucciones de unas pocas líneas en la pantalla y, por último, se encamina hacia la cocina. Usted le desenchufa el ordenador y enciende el televisor, sabiendo que la espera será larga. De nada vale llegar tarde a la cena, porque él no empieza hasta que no nos presentamos en su casa, pues para él parte de la fiesta consiste en mostrarte el ordenador en funcionamiento. Es evidente que Juan no usa eficazmente su ordenador.





Roy Ingram

## Los discos adecuados

Rosa María siempre había tenido una enorme colección de discos y se las había ingeniado para sacarle el máximo partido. La experiencia le había enseñado que es esencial seleccionar los discos adecuados para cada tipo de reunión. Recientemente Rosa decidió comprarse un ordenador y, por razones de velocidad y de capacidad de almacenamiento, escogió un sistema con dos discos gemelos. Entonces confeccionó sobre el papel los encabezamientos apropiados para las categorías de música que ella deseaba incluir en su programa de base de datos. Había dos grupos principales, cada uno de ellos con subtítulos que a su vez se volvían a subdividir. La división principal era entre música "negra" y música "blanca". La música negra consistía en tres subgrupos principales: reggae, jazz y soul. La música blanca se dividía en rock y nueva ola. El rock blanco se subdividía en *heavy metal* y rock progresivo, y así sucesivamente. Rosa utilizó el gestor de la base de datos para escoger discos según la ocasión (dando, por ejemplo, la orden de listar todos los discos de rock y luego haciendo una selección de nivel inferior de *heavy metal* o bien rock progresivo). Habiendo inspeccionado la lista de los títulos de discos visualizados en la pantalla, hizo un listado por impresora de los títulos elegidos y luego fue a la estantería y cogió los discos que deseaba para esa ocasión.

base de datos, llamar al archivo y encontrar el número de Lupita Pérez, podría realizar seis llamadas telefónicas a números obtenidos de su agenda o de un archivo de fichas de cartera. Se podría pensar en guardar los nombres y números de teléfono en un archivo con alguna otra finalidad, como imprimir cartas y etiquetas estándar, pero utilizar un ordenador cuando los sistemas manuales ordinarios resultan adecuados no tiene el menor sentido.

Un paquete de base de datos más ambicioso ofrecerá todas estas facilidades y proporcionará además la capacidad de utilizar la aritmética para obtener datos tales como la suma total de beneficios obtenidos mediante las cuotas del club, o calcular cuál es el servicio del club que se emplea con mayor frecuencia. Se pueden enlazar varios archivos de modo que los datos de temas relacionados entre sí se puedan utilizar conjuntamente. Por ejemplo, un jugador determinado que pertenezca a nuestro club imaginario tiene sus detalles personales en un archivo y los detalles de su rendimiento en las competiciones del club en otro. Cuando se requiera un historial de su "expediente", los dos archivos se pueden emplear juntos. Para una "instantánea" del rendimiento de todos los socios del club en una fecha determinada, se puede acceder al archivo llamando a cada registro para una cierta fecha, en vez de llamar a cada registro para una cierta persona. Con el fin de ejecutar un programa que posee facilidades tan amplias, casi con seguridad será necesario utilizar un ordenador de sobremesa o de oficina con un mínimo de 64 Kbytes de memoria y unidades de disco.

En los últimos años se ha producido un notable aumento en el empleo de sistemas de información. Los médicos utilizan bases de datos para llevar los detalles del historial clínico de sus pacientes. Quienes trabajan en investigación las emplean para la clasificación especializada y referencias cruzadas de datos. Las empresas llevan listas de correspondencia e informaciones actualizada relativa a los clientes y siempre disponible. Cuanta más información hay disponible, más se la utiliza. En la actualidad las bases de datos pueden tratar cualquier clase de información, desde simples listados hasta complicados sistemas con archivos múltiples, facilidades para crear informes elaborados y la capacidad de procesar información de forma que se pueda transferir a otro software, como los procesadores de textos. La capacidad para comunicarse con bases de datos de acceso público como Prestel y Micronet 800 le proporciona al usuario de un ordenador personal acceso a una enorme cantidad de información. Ello señala el camino hacia un futuro en el que su micro se podría convertir en un terminal enlazado con un gran ordenador central (*mainframe*).

# Sprites submarinos

**Analizaremos ahora con detalle las rutinas que controlan el barco, el submarino y las cargas de profundidad de nuestro juego "Subhunter"**

Para situar un sprite en la pantalla hay que especificar una coordenada X y una Y. Cada sprite se define en un cuadriculado de 21 × 24 pixels y las coordenadas se miden desde la *esquina superior izquierda* del cuadriculado. Las coordenadas se guardan en registros del chip VIC. Éstas se asignan del siguiente modo:

| Número de sprite | 0 | | 1 | | 2 | | 3 | | 7 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Coordenada | X | Y | X | Y | X | Y | X | Y | X | Y |
| Registro VIC | V | V+1 | V+2 | V+3 | V+4 | V+5 | V+6 | V+7 | V+14 | V+15 |

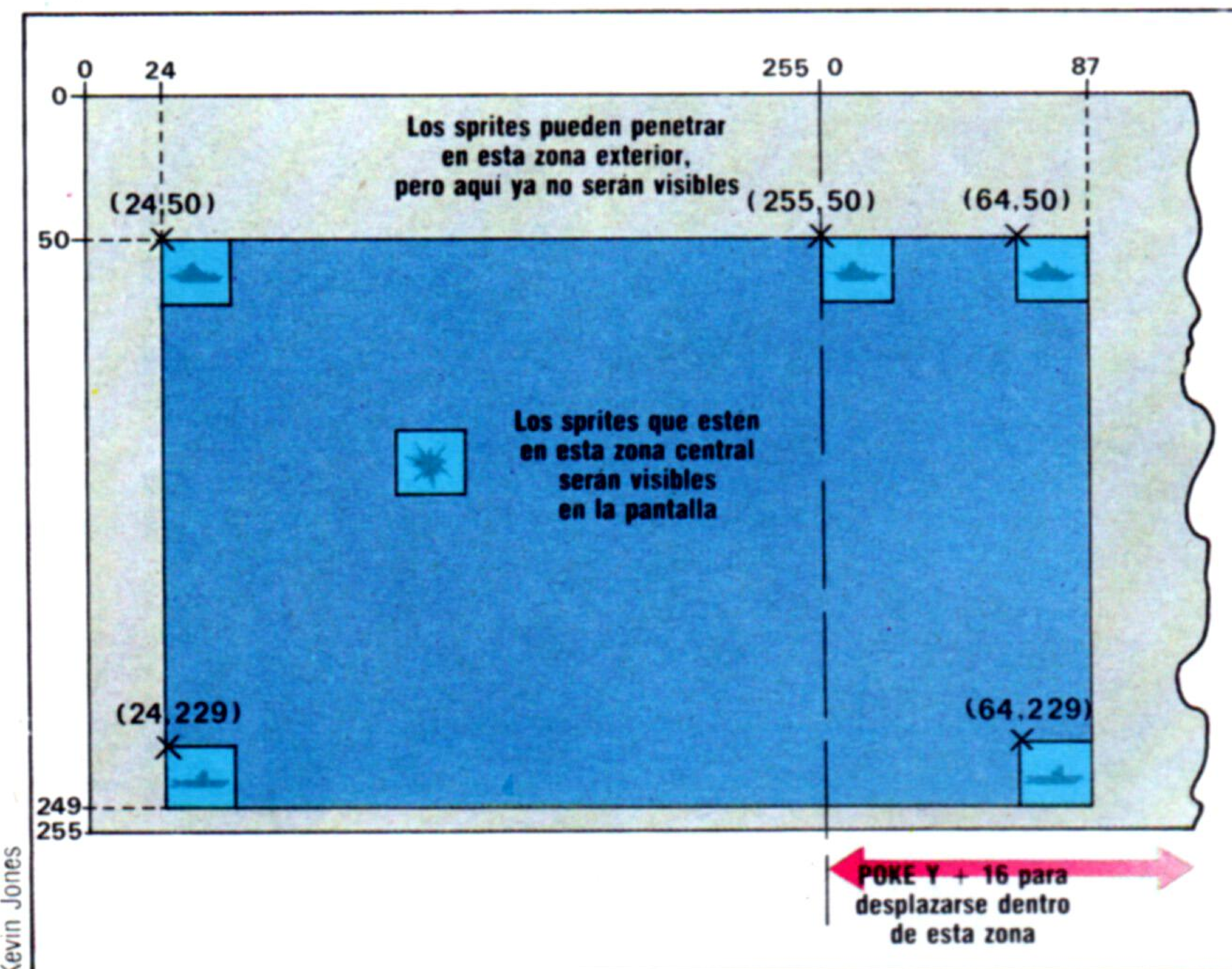
Cada posición puede aceptar números comprendidos en la escala 0 a 255. Esto es más que suficiente para especificar uno de los 200 pixels en la dirección vertical (Y), y la capacidad restante se utiliza para permitir que el sprite aparezca o desaparezca por la parte superior o inferior de la pantalla. Sin embargo, los 320 pixels de la dirección horizontal (X) superan el máximo de 255 que permite un registro de ocho bits, de modo que a cada sprite se le destina otro bit. Estos bits extras se agrupan juntos en un único registro con la dirección V+16. El diagrama inferior muestra los límites de la pantalla para sprites totalmente visibles y sin ampliar.

El movimiento de la embarcación se controla mediante las líneas del programa 230-250 y 270-290. Las coordenadas iniciales para el barco se establecieron en la línea 2270 de la rutina de creación de sprites (véase p. 745). El barco sólo se desplazará en dirección horizontal, de modo que la coordenada Y permanecerá fija. Si ésta se pone a 80, el barco estará correctamente situado en el océano, que se diseñó mediante la rutina de preparación de la pantalla. La coordenada X del barco, X0, se pone inicialmente en 160, que proporciona una posición de comienzo centrada en la pantalla.

El barco se controla mediante el teclado, utilizando las teclas "Z" y "X" para el movimiento a izquierda y derecha. Las líneas 230-250 del bucle principal obtiene (GET) un carácter pulsado en el teclado. Si no ha sido pulsada ninguna tecla, la ejecución del programa continúa (al contrario que la orden INPUT, que interrumpe un programa hasta que se pulsa una tecla). El valor obtenido por GET se almacena luego en A$ y se emplea para modificar la coordenada X del barco: si se pulsa la tecla "Z", el barco se mueve una pequeña distancia hacia la izquierda y es decrementada la coordenada X; si se pulsa la tecla "X" el barco se desplazará la misma distancia hacia la derecha y la coordenada X será incrementada. La segunda parte de las líneas 240 y 250 contiene una condición para ver si el barco ha alcanzado los límites de su viaje. La estructura IF...THEN del BASIC Commodore es tal que si la primera condición de una línea de programa no se cumple, el resto de la línea no se ejecuta. En nuestro programa, la verificación de la condición está dispuesta de modo que el ordenador no pierda tiempo en cálculos innecesarios.

El movimiento del submarino está controlado por las líneas 300-350 y 2500-2570 y opera dentro del bucle principal del programa, tal como refleja el diagrama de flujo contiguo. La subrutina "restaurar coordenadas submarino" que empieza en la línea 2500 utiliza la función RND para seleccionar la profundidad del submarino, Y3, y la velocidad, DX. Y3 siempre será un número entero comprendido entre 110 y 214, lo que asegura que el sumergible no aparezca por encima de la superficie del mar o por debajo del lecho marino. El factor de velocidad, DX, varía entre 1 y 3 y determina la cantidad de pixels con la que se incrementará o decrementará la coordenada X del submarino. La coordenada X del submarino y el bit de registro V+16 que controla las coordenadas X del sprite 3 por encima de 255 están ambas puestas a cero.

En las líneas 300-350 del bucle principal se le suma el valor seleccionado de DX a la coordenada X del submarino y ésta se verifica entonces para ver si éste ha alcanzado el borde de la pantalla. Las líneas 340 y 350 nos permiten tratar el problema de tener valores de X3 que superen 255. Cuando la coordenada X aumenta a, supongamos, 256, deben suceder dos cosas: el bit correspondiente al sprite 3 en el registro V+16 se pone a uno y el registro normal




**La posición de los sprites**
Este diagrama refleja los parámetros para situar los sprites en la pantalla del Commodore 64. Un sprite situado dentro de la parte central de esta pantalla será visible en la pantalla. Si el sprite se desplaza hasta la parte exterior de la pantalla, entonces no aparecerá a la vista. Los cuatro sprites de la esquina de la zona central muestran los límites hasta dónde se puede mover un sprite y ser todavía completamente visible. Se indican las coordenadas de las esquinas superiores izquierda de los sprites

Kevin Jones



de la coordenada X vuelve a empezar desde cero. La siguiente tabla muestra lo que sucede en los registros a medida que el submarino cruza la frontera de X = 255:

| X3 | V+16 | | | | | | | | V+6 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 254 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 255 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 256 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 257 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

La variable H3 se pondrá a uno si el valor de X3 sobrepasa 255. De modo similar, L3 se pondrá de nuevo a cero si H3 se convierte en uno. Los valores de L3 y H3 se pueden colocar entonces (POKE) en los registros V+6 y V+16.

Durante el juego pueden arrojarse cargas de profundidad en cualquier momento. Para hacer un programa directo, obedeceremos la regla de que después de que se haya lanzando una carga no se podrá disparar ninguna otra hasta que:

a) se haya acertado al submarino; o

b) las cargas hayan rozado el submarino; o

c) las cargas no hayan dado en el submarino y hayan llegado al lecho marino.

El bucle principal del programa tiene que realizar dos tareas con respecto a las cargas de profundidad: debe detectar la pulsación de la tecla "M" y, después de disparada la carga de profundidad, debe controlar su movimiento vertical. Asimismo, el programa debe asegurar que no se disparen nuevas cargas de profundidad mientras haya alguna en proceso de caída. Este último problema se puede resolver mediante la utilización de un indicador. Ésta es una técnica que se suele aplicar con frecuencia en el control de programas, señalando que un acontecimiento determinado se ha producido o no. En nuestro programa vamos a emplear la variable FL para indicar el disparo de una carga de profundidad. Su valor será uno si hay una carga cayendo, de lo contrario será cero. En la línea 100 del programa el valor de FL se pone inicialmente a cero. La línea 260 accede a la subrutina "Preparar cargas de profundidad" en la línea 3000 si se pulsa "M" y el indicador está en cero. En la línea 400 se utiliza una segunda subrutina para mover un sprite de carga de profundidad disparada, y a éste se accede por la línea 380.

La subrutina "preparar cargas de profundidad" tiene que realizar tres funciones:

1) Poner el indicador FL a uno como señal de que se ha arrojado una carga de profundidad.

2) Establecer las coordenadas de comienzo: la coordenada X toma su valor a partir de la del barco y la coordenada Y se pone inicialmente en 95, situando la carga bajo la superficie del mar.

3) Encender el sprite de la carga.

La subrutina "mover carga de profundidad" se utiliza para mover la carga de profundidad hacia abajo por la pantalla. Además, se han de verificar estas condiciones, si:

1) La carga de profundidad ha pasado junto al submarino *o* (OR) alcanzado el fondo.

2) La carga de profundidad ha hecho blanco en el submarino.

Si se ha producido lo primero, se puede apagar el sprite de la carga de profundidad y restablecer el indicador a cero, permitiendo arrojar otra carga de profundidad. El segundo acontecimiento se verifica

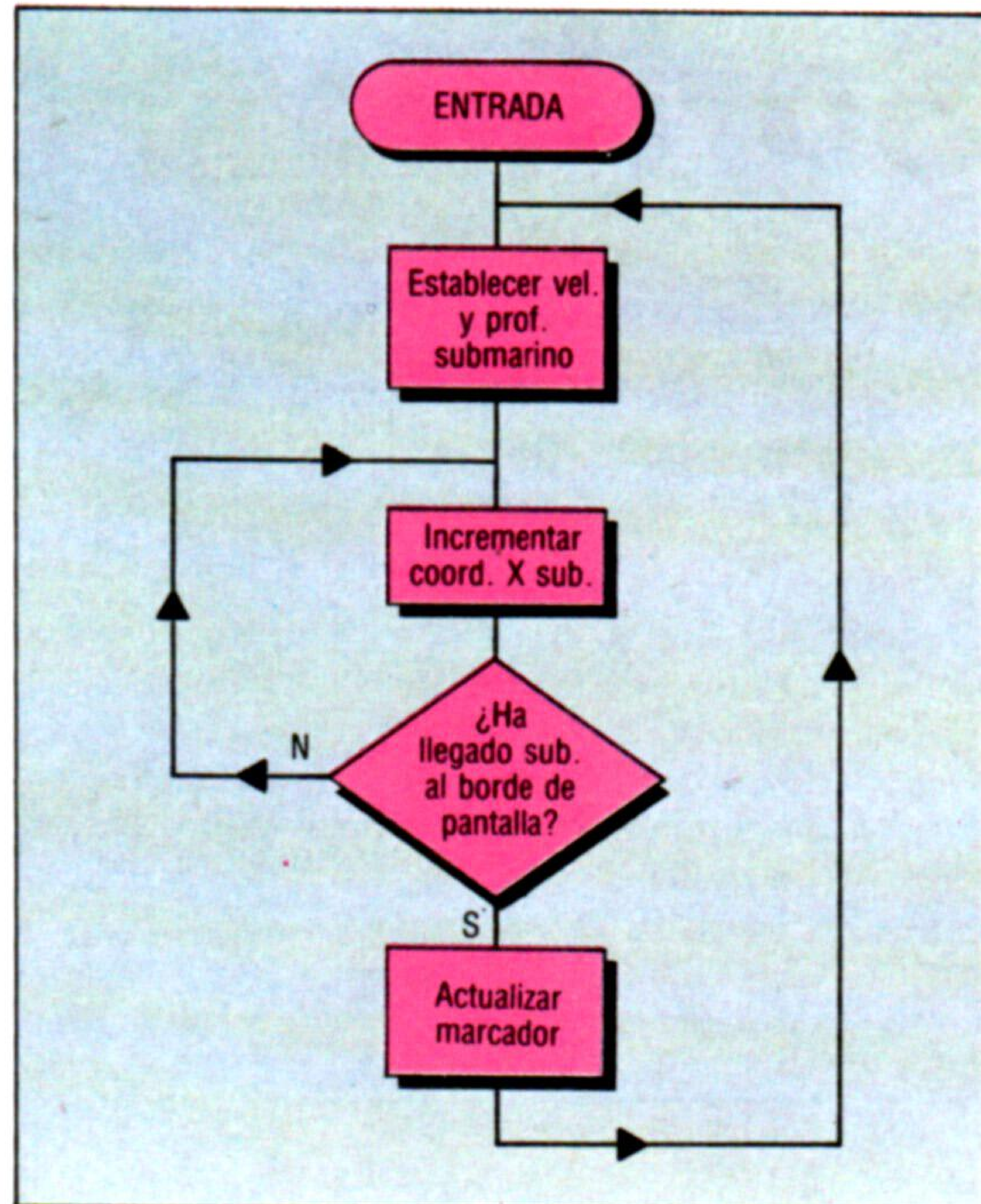



**Diagrama de flujo subacuático**
Este sencillo diagrama de flujo muestra cómo el programa controla el movimiento del sprite del submarino. Selecciona al azar una profundidad y una velocidad, asegurándose de que el sumergible esté por debajo de la superficie y por encima del fondo. El submarino se desplaza entonces de forma uniforme a través de la pantalla hasta llegar al otro lado

utilizando otra característica de los sprites del Commodore 64: el registro de colisión de sprites. Al igual que otros registros del chip VIC, este registro, V+30, posee un bit correspondiente a cada sprite. Si un sprite determinado se ve implicado en una colisión con otro sprite, entonces el bit correspondiente de este registro se pone a uno. Por consiguiente, si el submarino (sprite 3) y la carga de profundidad (sprite 2) colisionan, el contenido del registro V+30 será 12 (00001100 = 12). Tomando (PEEK) este registro y verificando su contenido podemos saber si la carga de profundidad ha hecho blanco en el submarino. Si así fuera, en la línea 5000 se accedería a otra subrutina "HIT" (blanco). De esta subrutina nos ocuparemos en el capítulo final del proyecto, así como de las instrucciones para actualizar el MARCADOR MAX y volver a comenzar el juego.

# Subrutinas de movimiento del Subhunter

```
130  GOSUB 2500:REM ESTABLECER COORDENADAS SUB
230  GET A$
240  IF A$="Z" THEN X0=X0-1.5:IF X0<24 THEN X0=24
250  IF A$="X" THEN X0=X0-1.5:IF X0>245 THEN X0=245
260  IF A$="M" AND FL=0 THEN GOSUB 3000:REM PREPARAR
     CARGAS PROFUNDIDAD
265  :
270  REM ** MOVER BARCO **
290  POKE V,X0
295  :
300  REM ** MOVER SUB **
310  X3=X3+DX
315  :
320  REM ** SUB EN FIN PANTALLA? **
330  IF X3>360 THEN DS=1:GOSUB 5500:GOSUB 2500
340  H3=INT(X3/256):L3=X3-256 * H3
350  POKE V+6,L3:POKE V+16,PEEK(V+16)OR(8*H3)
360  IF FL=1 THEN GOSUB 4000:REM MOVER CARGA PROFUNDIDAD
370  GOTO 200:REM REITERAR BUCLE PRINCIPAL
380  :
390  :
2500 REM **** RESTAURAR COORDENADAS SUB ****
2510 Y3=110+INT(RND(TI)*105)
2520 X3=0:DX=RND(TI)*3+1
2530 POKE V+7,Y3:POKE V+6,X3
2540 POKE V+16,PEEK(V+16) AND 247
2550 RETURN
2560 :
2570 :
3000 REM *** PREPARAR CARGAS PROFUNDIDAD ***
3010 :
3020 REM ** PONER INDICADOR **
3030 FL=1
3040 :
3050 REM ** ESTABLECER COORDENADAS **
3060 Y2=95:X2=X0
3070 POKE V+4,X2:POKE V+5,Y2
3080 :
3090 REM ** ENCENDER SPRITE 2 **
3100 POKE V+21,PEEK(V+21)OR4
3110 RETURN
3120 :
3130 :
4000 REM ** MOVER CARGA PROFUNDIDAD **
4010 :
4020 REM ** INCREMENTAR COORD Y **
4030 Y2=Y2+2
4040 POKE V+5,Y2
4050 :
4060 REM ** VERIFICAR FONDO Y APAGAR **
4070 IF Y2>Y3+25 OR Y2>216 THEN POKE V+21,PEEK(V+21) AND
     251: FL=0
4080 :
4090 REM ** VERIFICAR SI SUBMARINO ACERTADO **
4100 IF PEEK (V+30)=12 THEN GOSUB 5000:REM RUTINA HIT
4110 RETURN
4120 :
4130 :
```

# Sortilegio de la suma

**Presentamos un rompecabezas matemático, el "cuadrado mágico", cuya implementación en ordenadores personales resulta muy interesante**

Un cuadrado mágico es un cuadriculado de celdas en las que se introducen números enteros positivos (1, 2, 3, 4, 5, etc.), menos el cero, sin que se repita ninguno de ellos. El objetivo de este ejercicio consiste en distribuir tales números de forma que al sumarlos en cada columna y cada fila den el mismo resultado. El cuadrado mágico más simple es una caja de 3x3, y una posible solución es la que ofrecemos a continuación:

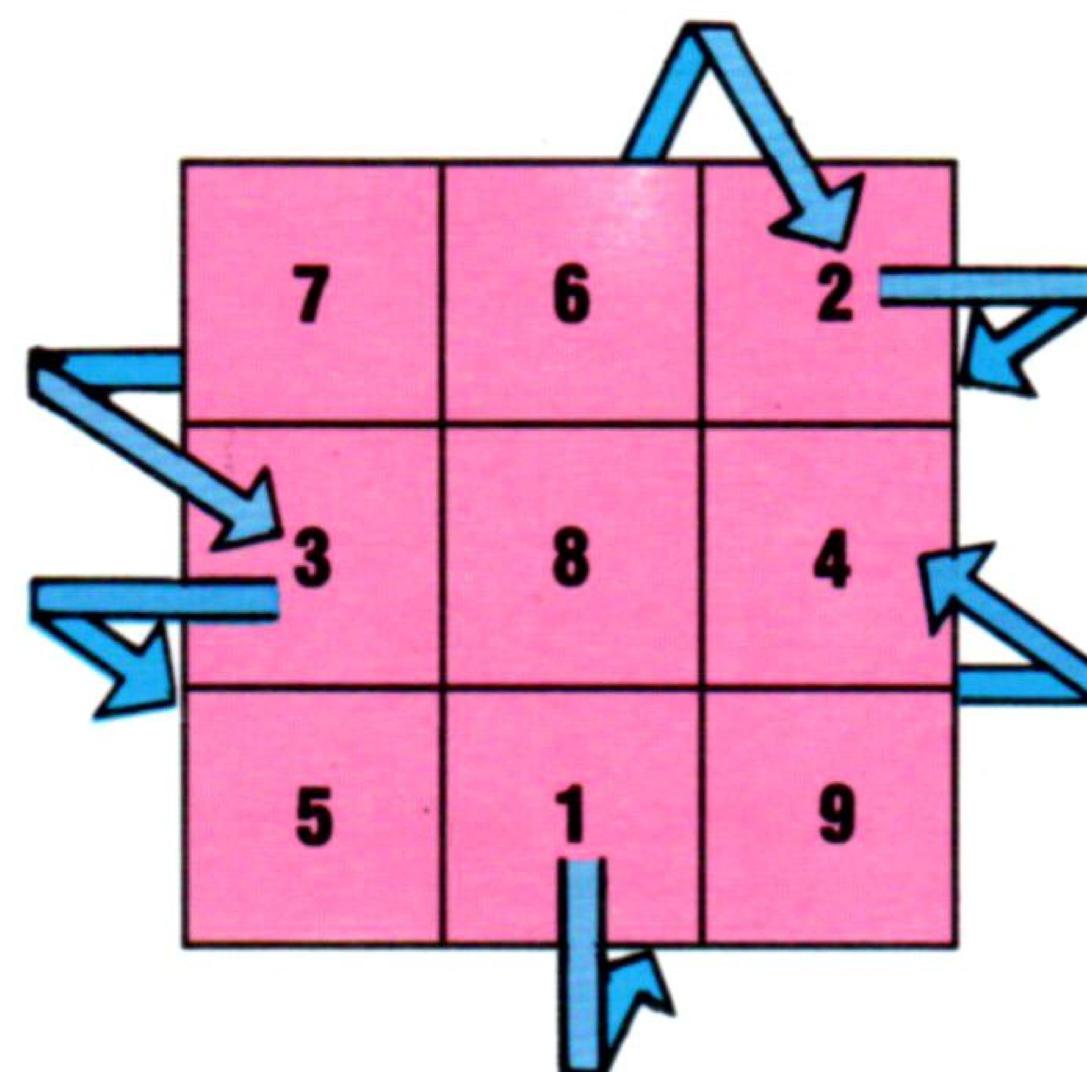
| | | | |
|---|---|---|---|
| 7 | 6 | 2 | 15 |
| 3 | 8 | 4 | 15 |
| 5 | 1 | 9 | 15 |
| 15 | 15 | 15 | |

En este ejemplo, todas las fichas y columnas suman 15. A modo de ejercicio, pruebe a construir usted mismo un cuadrado, pero esta vez utilice uno de 5x5 y use los números del 1 al 25. Descubrirá que no es tán fácil como parece, y de seguro le obligará a un buen número de ensayos erróneos antes de obtener la solución correcta.

Pero su ordenador puede hacer que las cosas resulten más sencillas. Una solución es escribir un programa para rellenar los cuadrados con números y luego verificar la suma de cada fila y cada columna. Sin embargo, éste método podría tardar varias horas en hallar la solución incluso para cuadrados bastante pequeños, de siete o nueve celdas de anchura. Lo que hace falta es un procedimiento para generar los números. Para ahorrarle el esfuerzo de descubrir su propio método, he aquí uno que funciona siempre:

1) Empezar con el número 1 en la celda central de la fila anterior.
2) Después de llenar cada celda, desplazarse una celda hacia abajo y una a la derecha antes de poner el número siguiente. Si al desplazarse hacia la derecha se sale del borde derecho del cuadriculado, vaya a la primera columna. Del mismo modo, si al desplazarse hacia abajo se sale de la parte inferior de la rejilla, pase entonces a la fila superior.
3) Si la celda que toca llenar ya está ocupada, desplazarse una posición hacia la izquierda de la última celda utilizada.
4) Continuar así hasta llenar todas las celdas.

La aplicación de este procedimiento a nuestro ejemplo nos muestra cómo se genera un cuadrado de 3x3·




Es muy fácil escribir un programa que haga esto para un cuadrado de cualquier tamaño. Tan sólo hay que crear una matriz bidimensional vacía de las dimensiones correctas para el cuadrado mágico deseado y luego construir un bucle para rellenarla de acuerdo a las reglas dadas. Observará que el méto-

## Cuadrados encantados

Estos cuadrados se generaron con el programa y éste los ha verificado con éxito. En el cuadrado de 15 × 15 se puede observar una configuración en diagonal de números de tres y dos dígitos, que es consecuencia del algoritmo usado para su construcción

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 52 | 42 | 32 | 22 | 12 | 2 | 73 | 72 | 62 |
| 63 | 53 | 43 | 33 | 23 | 13 | 3 | 74 | 64 |
| 65 | 55 | 54 | 44 | 34 | 24 | 14 | 4 | 75 |
| 76 | 66 | 56 | 46 | 45 | 35 | 25 | 15 | 5 |
| 6 | 77 | 67 | 57 | 47 | 37 | 36 | 26 | 16 |
| 17 | 7 | 78 | 68 | 58 | 48 | 38 | 28 | 27 |
| 19 | 18 | 8 | 79 | 69 | 59 | 49 | 39 | 29 |
| 30 | 20 | 10 | 9 | 80 | 70 | 60 | 50 | 40 |
| 41 | 31 | 21 | 11 | 1 | 81 | 71 | 61 | 51 |

Ian McKinnell



do sólo funciona para cuadrados mágicos con una cantidad impar de celdas; su programa deberá rechazar los cuadriculados de número par.

Hay que tener cuidado cuando se visualiza el cuadrado. En aras de la pulcritud, todos los números se deben alinear correctamente en filas y columnas. Esto es bastante fácil de conseguir si su micro posee la orden PRINT USING. De no ser así, es mejor convertir el número a imprimir en una serie. Ésta se puede entonces rellenar con caracteres espacio de modo que el "número" siempre tenga la misma longitud. Una subrutina para conseguirlo es:

```
1000  REM Convertir A en A$ y alinear
1010  A$=STR$(A)
1020  IF LEN(A$)<3 THEN A$=" "+A$:GOTO
1020
1030  RETURN
```

El método exacto dependerá, por supuesto, del ordenador que se esté utilizando.

El siguiente problema es el tamaño de la pantalla; la mayoría de los micros son incapaces de visualizar en pantalla cuadrados mágicos grandes. Una pantalla de 40 columnas tiene espacio para 13 columnas de dos dígitos, pero un cuadrado de 13x13 incluirá números de tres dígitos, de modo que el más grande que se puede visualizar es un cuadrado de 9x9. Una impresora permitirá generar cuadrados mucho más grandes. La mayoría de las impresoras tienen una anchura máxima de 80 o 132 columnas y se pueden imprimir por secciones cuadrados más grandes que pueden ser unidos posteriormente.

El objetivo global de este proyecto es crear el cuadrado mágico más grande que se pueda y presentarlo con la mayor pulcritud posible. Como experimento, intente escribir un programa de ensayo y error y determine cuánto tarda en hallar la respuesta (aunque conseguir un resultado correcto le ocupará bastante tiempo). O intente buscar procedimientos aún más rápidos para generar los cuadrados.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 130 | 114 | 98 | 82 | 66 | 50 | 34 | 18 | 2 | 211 | 210 | 194 | 178 | 162 | 146 |
| 147 | 131 | 115 | 99 | 83 | 67 | 51 | 35 | 19 | 3 | 212 | 196 | 195 | 179 | 163 |
| 164 | 148 | 132 | 116 | 100 | 84 | 68 | 52 | 36 | 20 | 4 | 213 | 197 | 181 | 180 |
| 166 | 165 | 149 | 133 | 117 | 101 | 85 | 69 | 53 | 37 | 21 | 5 | 214 | 198 | 182 |
| 183 | 167 | 151 | 150 | 134 | 118 | 102 | 86 | 70 | 54 | 38 | 22 | 6 | 215 | 199 |
| 200 | 184 | 168 | 152 | 136 | 135 | 119 | 103 | 87 | 71 | 55 | 39 | 23 | 7 | 216 |
| 217 | 201 | 185 | 169 | 153 | 137 | 121 | 120 | 104 | 88 | 72 | 56 | 40 | 24 | 8 |
| 9 | 218 | 202 | 186 | 170 | 154 | 138 | 122 | 106 | 105 | 89 | 73 | 57 | 41 | 25 |
| 26 | 10 | 219 | 203 | 187 | 171 | 155 | 139 | 123 | 107 | 91 | 90 | 74 | 58 | 42 |
| 43 | 27 | 11 | 220 | 204 | 188 | 172 | 156 | 140 | 124 | 108 | 92 | 76 | 75 | 59 |
| 60 | 44 | 28 | 12 | 221 | 205 | 189 | 173 | 157 | 141 | 125 | 109 | 93 | 77 | 61 |
| 62 | 46 | 45 | 29 | 13 | 222 | 206 | 190 | 174 | 158 | 142 | 126 | 110 | 94 | 78 |
| 79 | 63 | 47 | 31 | 30 | 14 | 223 | 207 | 191 | 175 | 159 | 143 | 127 | 111 | 95 |
| 96 | 80 | 64 | 48 | 32 | 16 | 15 | 224 | 208 | 192 | 176 | 160 | 144 | 128 | 112 |
| 113 | 97 | 81 | 65 | 49 | 33 | 17 | 1 | 225 | 209 | 193 | 177 | 161 | 145 | 129 |

```
10  REM***************
15  REM***CUADRADOS MAGICOS *****
20  REM******PREPARACION****
30  M=19:DIM A(M,M)
40  PRINT:PRINT"Cuadrados mágicos"
50  PRINT:PRINT"Cuantas filas?(1 a 19)";INPUT S
60  IF S<0 OR S<> INT(S) THEN PRINT"ERROR":GOTO 50
70  IF S>M THEN PRINT"ERROR":GOTO 50
80  IF S/2=INT(S/2) THEN PRINT"ERROR - Solo numeros impares":GOTO 50
90  REM****GENERAR CUADRADO****
100 X=INT(S/2)+1:Y=S:C=1
110 A(X,Y)=C
120 C=C+1:IF C>S*S THEN GOTO 200
130 X=X+1:IF X>S THEN X=1
140 Y=Y+1:IF Y>THEN Y=1
150 IF A(X,Y)<>0 THEN X=X-2:Y=Y-1
160 IF Y=0 THEN Y=S
170 IF X=0 THEN X=S
180 IF X=-1 THEN X=S-1
190 GOTO 110
200 REM****IMPRIMIR CUADRADO****
210 PRINT:PRINT
220 FOR Y=1 TO S:FOR X=1 TO S
230 A=A(X,Y):GOSUB 380:PRINT" ";A$;" ";
240 NEXT X:PRINT:NEXT Y

250 REM****VERIFICAR FILAS Y COLUMNAS*****
260 F=0
270 FOR Y=1 TO S:T=0
280 FOR X=1 TO S:T=T+A(X,Y):NEXT X
290 IF F=0 THEN U=T:F=1
300 IF T<>U THEN PRINT"ERROR - Fila 1 y fila";Y;"no coinciden":STOP
310 U=T:NEXT Y
320 FOR X=1 TO S:T=0
330 FOR Y=1 TO S:T=T+A(X,Y):NEXT Y
340 IF T<>U THEN PRINT"ERROR —Fila 1 y columna";X;"no
    coinciden":STOP
350 U=T:NEXT X
360 PRINT:PRINT"Todas las filas y columnas suman ";T
370 STOP
380 REM***CONV NUM-SERIE***
390 A$=STR$(A)
400 IF LEN(A$)<3 THEN A$=" "+A$:GOTO 400
410 RETURN
```



## Complementos al BASIC

Este programa está escrito en BASIC Microsoft, de modo que funcionará sin ninguna modificación en la mayor parte de los mismos. Ya saben los usuarios del Spectrum que deben insertar LET antes de todas las sentencias de asignación. El programa pide que se le suministre el número de filas (y, por lo tanto, de columnas) del cuadrado mágico, y verifica que éste sea un número impar, entero y positivo. Calcula y visualiza el cuadrado mágico y desde la línea 250 verifica su resultado.



# Números aleatorios

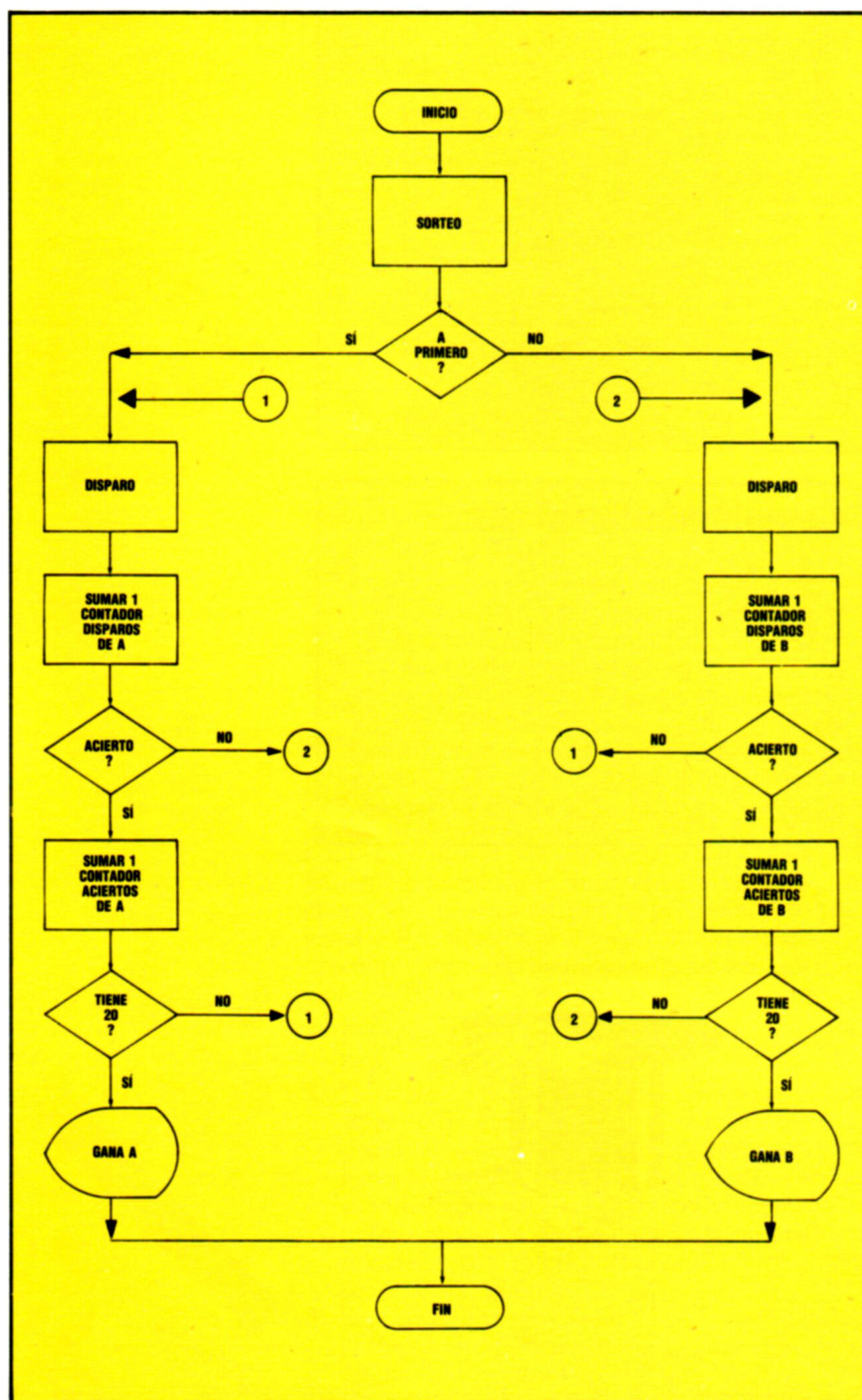
**El orden de intervención de los dos jugadores que citamos en nuestro ejemplo lo determina la función RND al generar un número aleatorio**

El programa siguiente es la transcripción al BASIC del ordinograma del capítulo anterior, con control del tiro al plato con contadores.

La parte que más difícil puede parecer a la hora de programar es la correspondiente al sorteo inicial. Este punto se ha resuelto con la utilización de la función RND, que permite generar números aleatorios. Éstos se crean mediante algoritmos que recurren a cálculos ya determinados y que dan una serie de números que parecen elegidos al azar.

Así, en el caso presente, se va a generar un número, el 1 o el 2 solamente, de modo que se pueda comparar el número y en función de él dar la señal del inicio al jugador A en caso de aparecer el 1 y al jugador B en caso de aparecer el 2 (sentencia 20). Igual proceso de comparación ocurre en las sentencias 50 y 110 cuando se trata de saber si se ha acertado el disparo (número 1) o se falló (número 2).

El programa se ha realizado para un Commodore 64; los contadores de disparos y los contadores de aciertos de uno y otro jugador se han denominado CA y CB, y A y B, respectivamente.

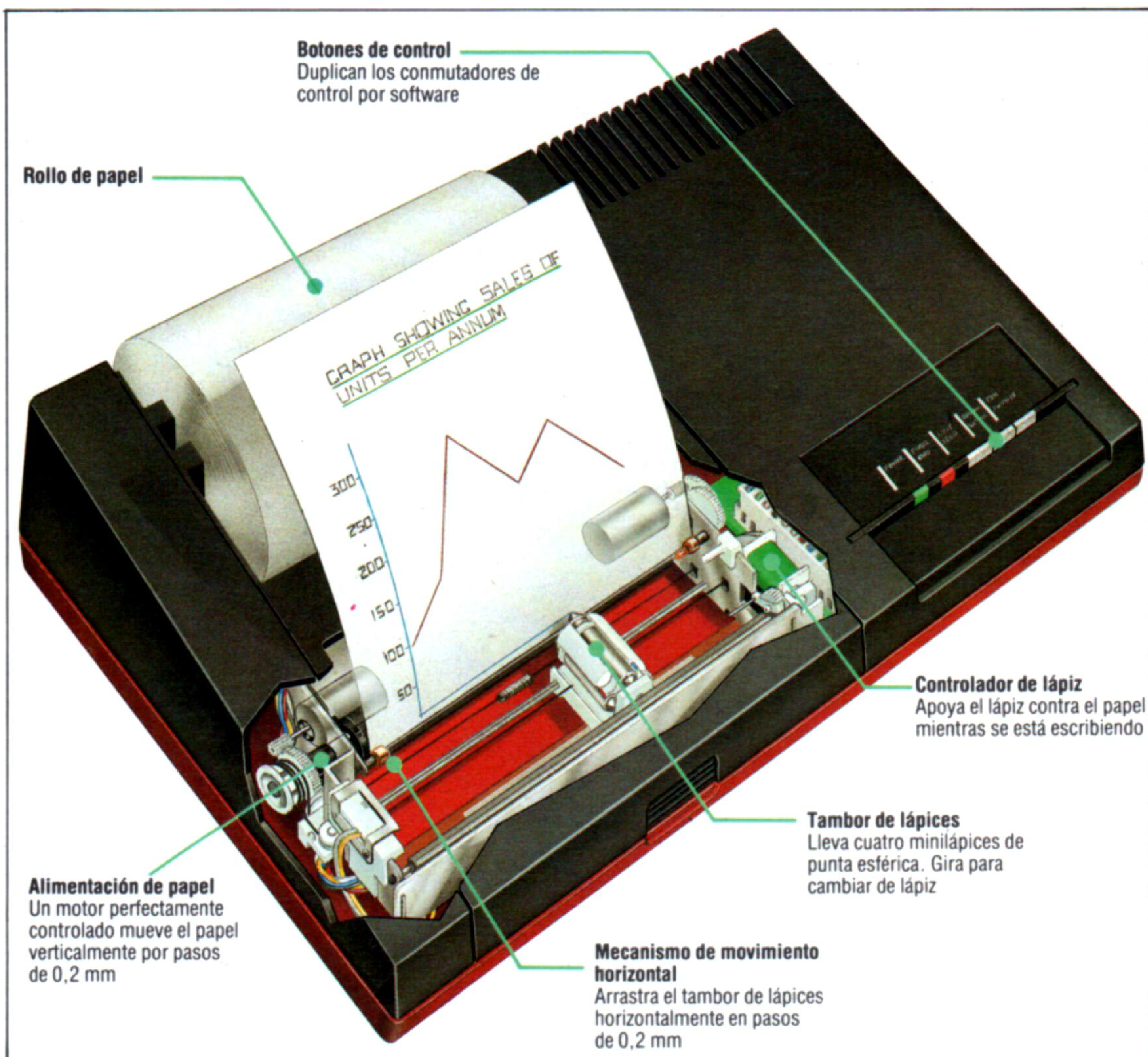



```
10  REM****TIRANDO AL PLATO****
15  REM****SORTEO*******
20  S=INT(RND(1)*2)+1
25  IF S=1 THEN PRINT"COMIENZA EL JUGADOR
    A" : GOTO 45
30  PRINT"COMIENZA EL JUGADOR B"
35  GOTO 100
40  REM*****DISPARANDO EL JUGADOR A
45  PRINT "DISPARA A"
50  X=INT(RND(1)*2)+1
55  CA=CA+1
60  REM****DECISION ACIERTO O FALLO
65  IF X=1 THEN GOTO 80
70  PRINT "FALLO"
75  GOTO 100
80  PRINT "ACIERTO"
83  A=A+1
85  IF A<>20 THEN GOTO 45
90  PRINT "GANADOR JUGADOR A EN" CA "
    DISPAROS"
95  END
99  REM*****DISPARANDO EL JUGADOR B
100  PRINT "DISPARA B"
110  R=INT(RND(1)*2)+1
120  CB=CB+1
130  REM****DECISION ACIERTO O FALLO
140  IF R=1 THEN GOTO 170
150  PRINT "FALLO"
160  GOTO 45
170  PRINT "ACIERTO"
175  B=B+1
180  IF B<>20 THEN GOTO 100
190  PRINT "GANADOR JUGADOR B EN" CB "
     DISPAROS"
200  END
```



# Plumas afines

**Veremos aquí un modelo de impresora-plotter que se comercializa bajo diversos nombres y que es especialmente adecuada para ser usada con micros**




Steve Cross

**Trazado de calidad**
La impresora-plotter tiene una resolución de alrededor de 500 ×2 000 pixels en modalidad plotter y una gama de tamaños. El color del lápiz se selecciona haciendo girar el tambor de lápices, y trazar o imprimir implica desplazar el tambor de los lápices horizontalmente a izquierda y derecha y enrollar y desenrollar el papel. El lápiz se puede apoyar contra el papel o apartarlo del mismo, para permitir el trazado o el movimiento sin huella, respectivamente. El mecanismo es capaz de realizar un trazado de gran precisión, pero el movimiento repetido del papel y el tambor produce una ligera pérdida de precisión

Atari 1020, Commodore 1510 y Oric MCP-40 son algunos de los nombres con los que se comercializa un plotter para microordenador notablemente barato. La parte interna de todas estas máquinas la fabrica una firma japonesa, adaptándola a las exigencias de cada empresa.

Además de su capacidad para gráficos, el plotter también puede imprimir textos, y por esa razón se lo conoce como *impresora-plotter*. Dentro de la unidad hay un cabezal rotatorio que contiene cuatro pequeños lápices de punta esférica. Para trazar una línea, el cabezal gira para seleccionar el color adecuado (los colores rojo, azul, verde y negro vienen como estándar) y luego el lápiz elegido se desplaza hasta hacer contacto con el papel. Una línea horizontal se traza moviendo el cabezal de un lado a otro; una línea vertical se traza mediante el movimiento hacia arriba y hacia abajo del papel. El texto se produce de forma muy similar a los gráficos; la impresora-plotter almacena los patrones para letras y otros caracteres en su propia memoria. Cuando se recibe una señal proveniente del ordenador, la impresora-plotter simplemente busca el carácter adecuado en su memoria interna y luego lo dibuja como si fuera un patrón de gráficos. La calidad de impresión resultante es excelente; por cierto, superior a la de la mayoría de las impresoras matriciales económicas.

Cuando está en modalidad de texto, al impresora-plotter opera exactamente igual que cualquier otra impresora. Aunque el papel que utiliza sólo tiene 115 mm de ancho, la unidad puede producir 40 u 80 caracteres por línea. La escasa anchura del papel hace que el texto de 80 caracteres quede más bien pequeño, pero la definición de los caracteres es muy buena. El texto se puede impri-

mir en cualquiera de los cuatro colores, con una velocidad de impresión de unos 10 caracteres por segundo. Esto es lento en comparación con la mayoría de las impresoras matriciales, pero resulta más o menos igual a la velocidad de las de rueda margarita.

Para producir gráficos, la impresora-plotter se coloca en modalidad de gráficos y entonces produce líneas, curvas y letras grandes. Si a la unidad se le envían caracteres estando en modalidad de gráficos, los interpretará como órdenes y no intentará imprimirlos. Por consiguiente, si necesitamos trazar líneas, emplearemos una orden en BASIC como la que ofrecemos:

```
LPRINT "D 0,100,100,100,100,0,0,0"
```

La D es la instrucción para dibujar líneas y los números que siguen dan las posiciones de los puntos a través de los cuales pasarán las líneas. En este caso (suponiendo que el cabezal de impresión esté posicionado en las coordenadas (0,0) al principio) se dibujará un cuadrado. Otras órdenes siguen un formato similar.

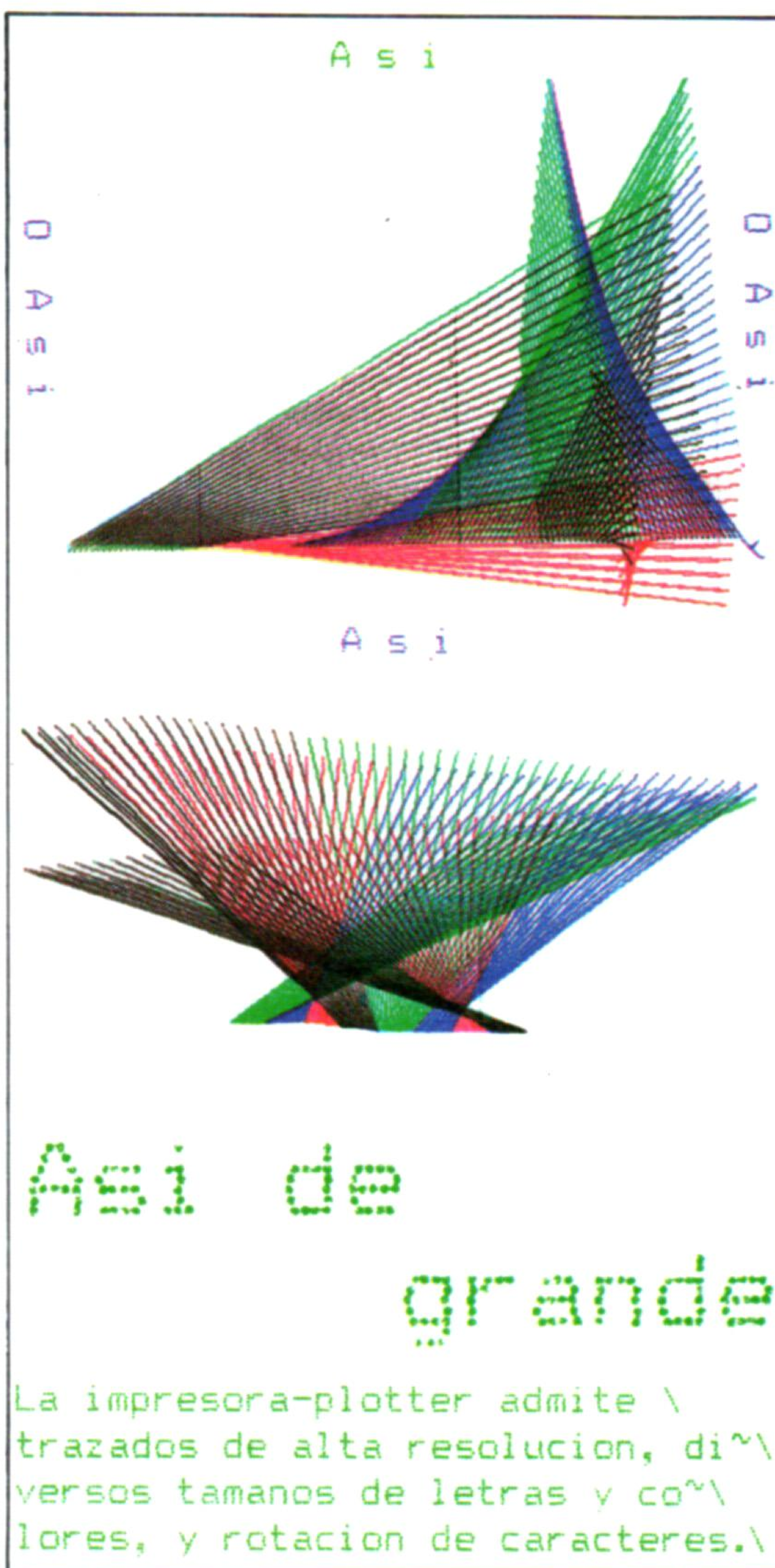
La anchura del papel se considera dividida en 480 pasos horizontales, que se utilizan para posicionar el cabezal de impresión. El papel empleado se suministra en un rollo de varios metros de longitud. Esto podría hacer suponer que no existen límites para la longitud vertical de un dibujo, pero en realidad el ligero resbalamiento del papel al moverlo verticalmente puede ocasionar que las líneas no encajen de la forma requerida, de modo que la impresora-plotter no permitirá que el papel se vuelva a enrollar más de una cierta distancia.

Aunque con la impresora-plotter se pueden realizar algunos gráficos muy complejos, escribir los programas para hacerlos lleva mucho tiempo; no hay ninguna forma sencilla de copiar una imagen directamente a partir de la pantalla del ordenador. El trazado de líneas es simple, pero la ausencia de una orden PAINT o FILL significa que sólo se pueden obtener bloques de color sólido trazando muchas líneas finas. Esto es lento y supone un desperdicio de tinta. El dispositivo tampoco puede dibujar en toda la anchura del papel, sino que hay que dejar un pequeño espacio vacío a cada lado. La unidad es especialmente apta para trazar gráficos y dispone de una orden que dibuja automáticamente los ejes de un gráfico, junto con las unidades correspondientes marcadas a intervalos elegidos.

En la modalidad de texto, la impresora-plotter produce dos tamaños de letra distintos. Sin embargo, utilizando la modalidad de gráficos se dispone de tamaños de texto mucho mayores y se permite imprimir letras de lado, pudiendo así imprimir mensajes largos en toda la longitud del papel. A los lápices de punta esférica se les acaba enseguida la tinta y es probable que se sequen si se los deja en el cabezal de impresión. Cada vez que la unidad se conecta, todos los lápices se prueban de forma automática, dibujando un cuadrado pequeño en cada color.

Los plotters profesionales son sumamente precisos y producen gráficos en color de una gran calidad. No sería razonable esperar que una unidad de este precio igualara estos niveles, pero hay que reconocer que los resultados conseguidos con la impresora-plotter tienen un nivel sorprendentemente bueno.






**Interfaces para plotter**
La tabla muestra las unidades que se pueden utilizar con los micros más usuales. El único problema será obtener el cable adecuado para conectarlos entre sí. Oric, Sharp, Commodore y Atari incluyen los cables necesarios para sus propios micros. Un ordenador popular que no está incluido en la lista es el Sinclair Spectrum, que carece de una interface adecuada

| | Atari 1020 | Commodore 1520 | MCP-40 | Oric MCP-40 | Sharp MZ-80P5(K) | Tandy CGP-115 |
|---|---|---|---|---|---|---|
| Atari 400/600/800 | ● | | | | | |
| BBC Micro | | | ● | ● | | ● |
| CGL/Sord M5 | | | ● | ● | | ● |
| Colour Genie | | | | | | ● |
| Commodore Vic-20 | | ● | | | | |
| Commodore 64 | | ● | | | | |
| Dragon 32/64 | | | ● | ● | | ● |
| Memotech MTX | | | ● | ● | | ● |
| Oric-1/Atmos | | | ● | ● | | ● |
| Sharp MZ-700 | | | | | ● | |
| Tandy TRS-80 Colour | | | | | | ● |

Chris Stevens

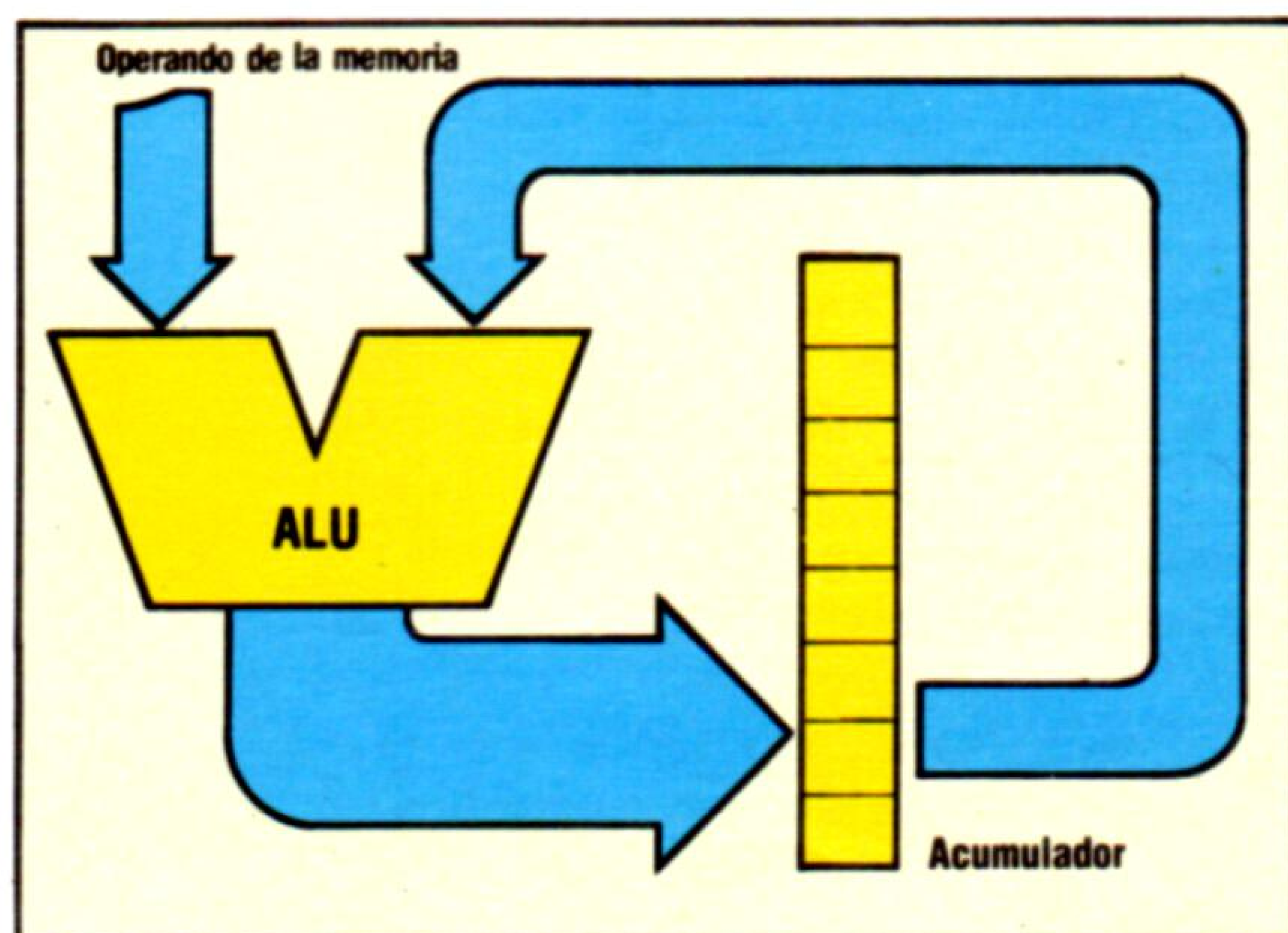
# Final lógico

## Concluimos esta serie estudiando la lógica de la CPU, y en concreto las funciones de la unidad aritmética lógica (ALU)

Son dos los tipos de funciones desempeñadas por la ALU: las *aritméticas* (suma, resta e incremento en una unidad), y las *lógicas* (que se resumen en tres: OR, AND y XOR).

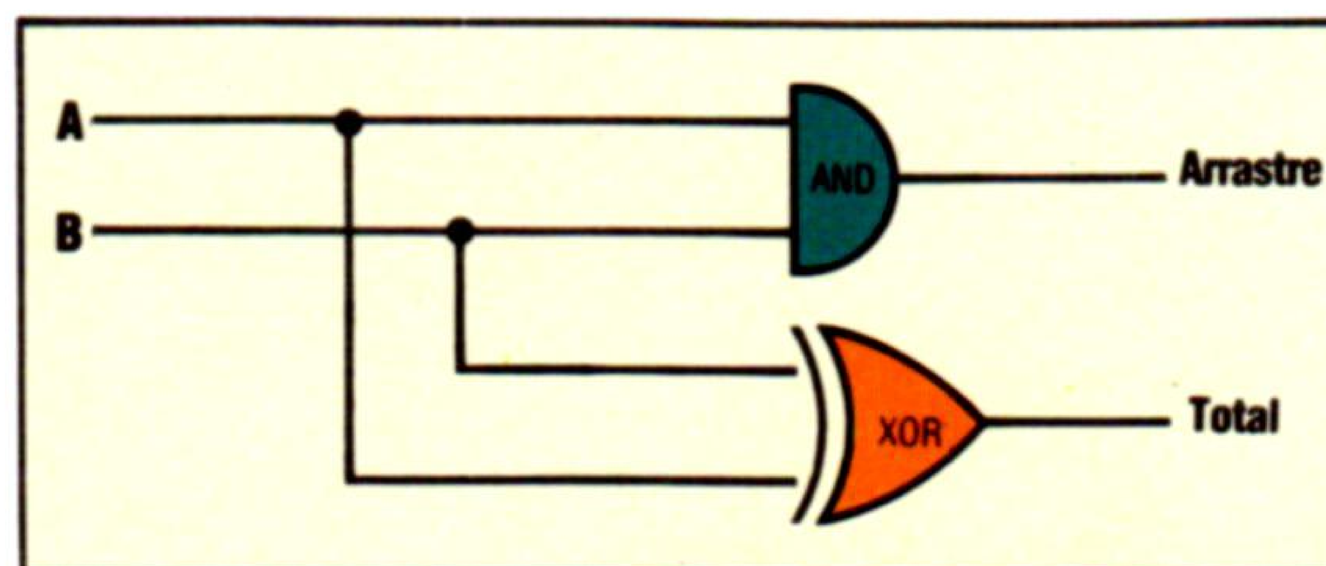
Funciones tales como la suma necesitan dos operandos (o sea, dos números con los que se realiza la operación). Otras, como el incremento, se bastan con un operando que se toma de un registro especial de la CPU llamado acumulador. Cuando son necesarios dos operandos, uno de ellos procede de la memoria principal. Ambos se moverán por los circuitos de la ALU donde tiene lugar la operación escogida. El resultado de la operación queda, finalmente, en el acumulador.

Veamos el recorrido de los datos a través del chip de la ALU en el siguiente dibujo:

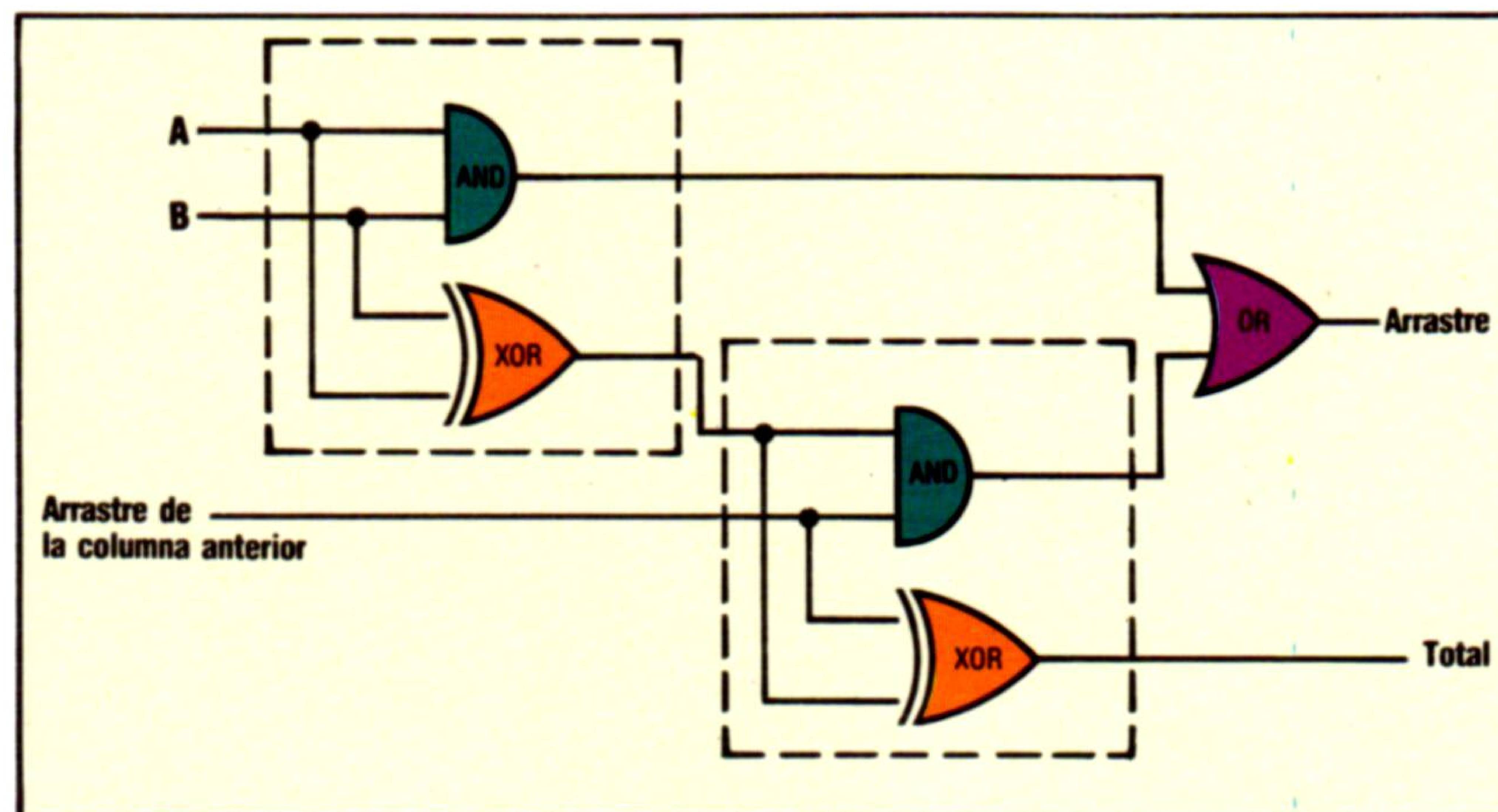



Los números, tanto en el acumulador como en la memoria, tienen una longitud de ocho bits. Estos bits que conforman los números circulan en paralelo por los circuitos de la ALU y la operación se realiza con los ocho bits a la vez. Entenderemos mejor los circuitos de la ALU si nos fijamos en uno solo de estos bits diseñando un circuito que realice con él las seis funciones mencionadas. Al bit del primer operando lo denominaremos A y al del segundo operando, B.

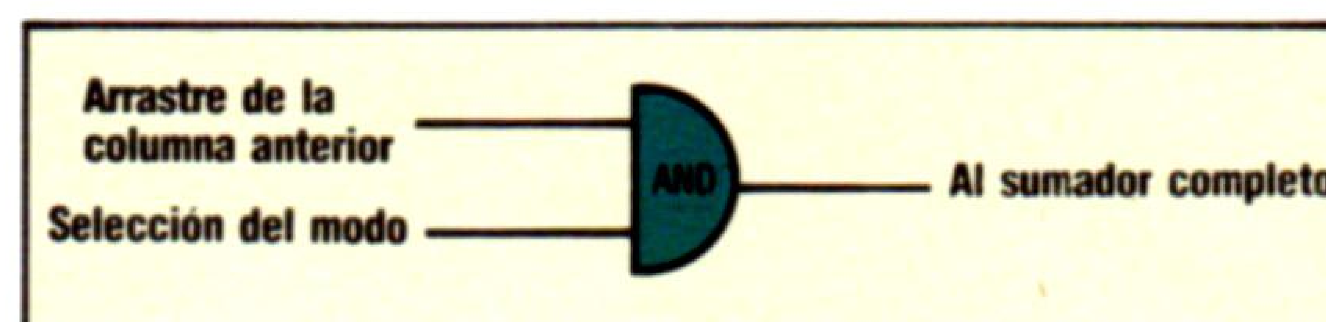
En la base de todo el dispositivo se encuentra el sumador completo. Ya lo dejamos diseñado en un capítulo anterior (véase p. 526), formado por dos circuitos sumadores incompletos que combinan las puertas AND, OR y NOT. Pero estos sumadores incompletos pueden simplificar su circuito usando la puerta XOR (o puerta OR exclusiva), como se ve en el primer diagrama de la columna derecha.




Acoplando dos sumadores incompletos, obtenemos un circuito sumador completo (segundo diagrama de esta columna).



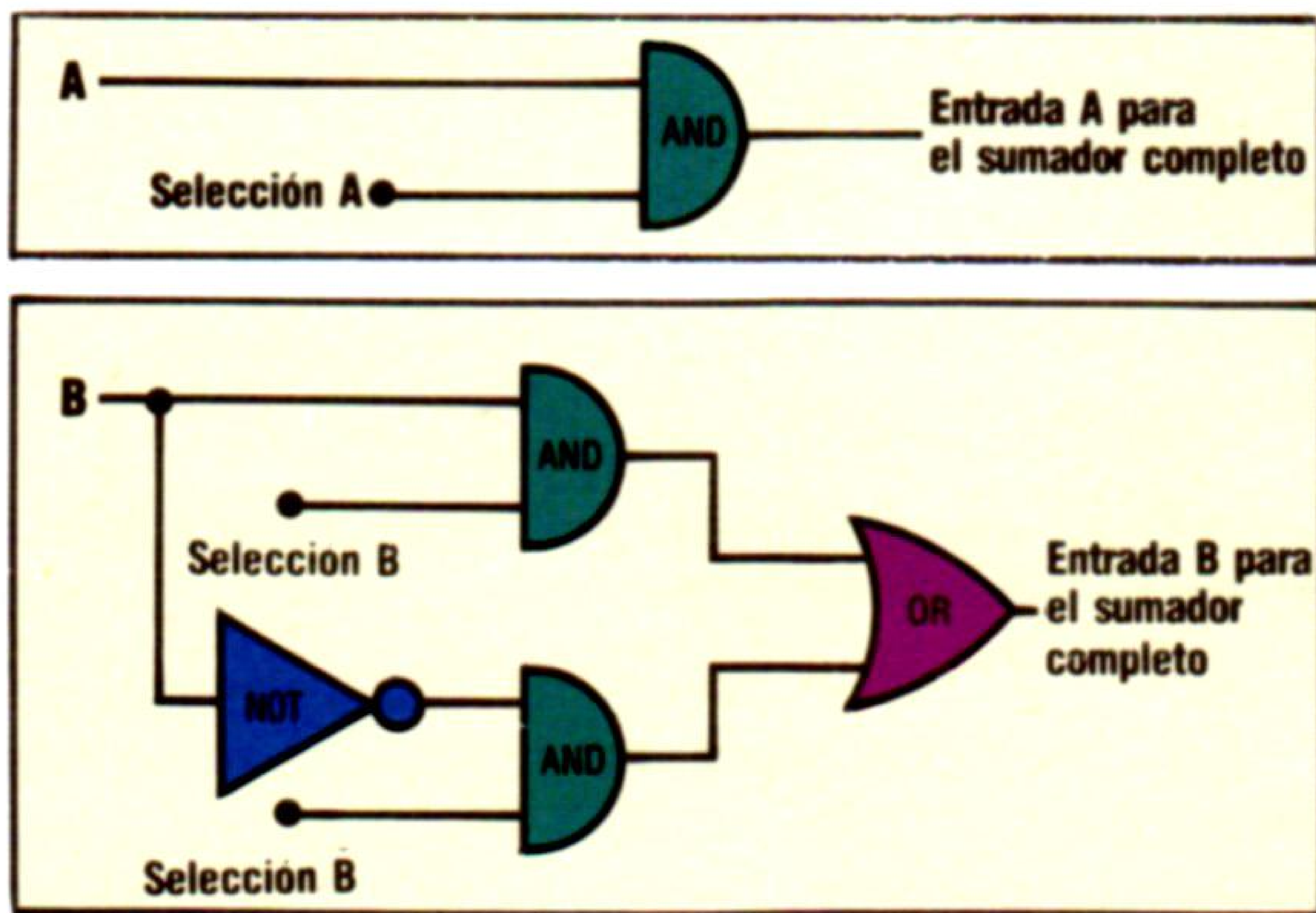

Vamos ahora a ver cómo añadiendo unos pocos circuitos, este sumador completo es capaz de realizar las restantes funciones de la ALU. Para ello estableceremos una serie de señales de control, la más importante de las cuales es la conocida como señal de *selección del modo*. Las operaciones aritméticas necesitan la entrada que llamamos "arrastre de la columna anterior", pero no así las operaciones lógicas. La señal de selección del modo conmutará esa entrada de arrastre y la pondrá en off o en on. Esto se logra con introducir una puerta AND:




Si ponemos en 1 la señal de selección del modo, la entrada del arrastre, necesaria en las operaciones aritméticas, se respeta gracias a la puerta AND. Para los casos en que el arrastre no es necesario, bastará poner a 0 esta señal. Como podemos añadir estas puertas AND en una u otra entrada, es posible seleccionar el bit A, el bit B o ambos.

En el proceso de resta utilizando la suma en complemento a dos, es necesario calcular el complemento a dos del sustraendo. Se deben cambiar, como dijimos, los unos en ceros, y los ceros en unos. Esto significa que debemos añadir a nuestro circuito sumador, si lo queremos emplear en las restas, algún otro circuito que seleccione la negación del bit B. Se trata de afectar la entrada B con una puerta NOT e incluir la señal de selección del modo empleando de nuevo la puerta AND. Tendríamos, para las entradas A y B, el diseño de circuitos de los dos primeros diagramas de p. 773.
Con estas cuatro señales de control es posible realizar cualquiera de las tres funciones aritméticas



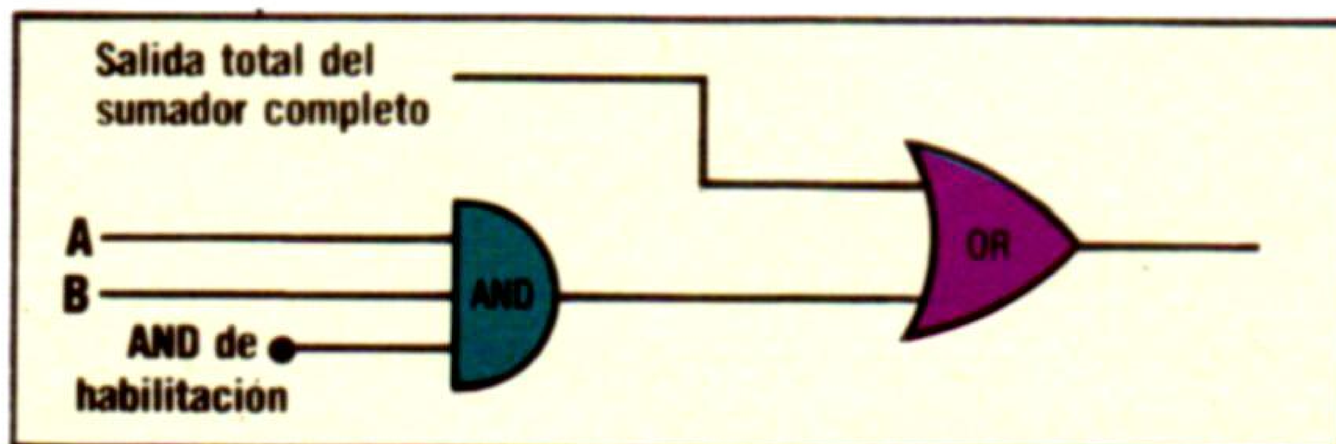



antes mencionadas. La tabla ilustra las combinaciones necesarias:

| Función | Selección modo | Selecc. A | Selecc. B | Selecc. $\bar{B}$ |
|---|---|---|---|---|
| Suma | 1 | 1 | 1 | 0 |
| Resta | 1 | 1 | 0 | 1 |
| Incremento de A (el 1.$^{er}$ arrastre se pone en 1) | 1 | 1 | 0 | 0 |
| Increm. de B | 1 | 0 | 1 | 0 |

Para las operaciones lógicas, ya que prescinden del "arrastre de la columna anterior", la señal de selección del modo puede ponerse en cero. Esto significa que para obtener la función lógica XOR debemos colocar en HI (alto) tanto la selección A como la selección B, y en LO (bajo) la selección del modo.

La función AND no se obtiene tan directamente, pues necesita las entradas A y B por separado, junto con una señal de selección con AND.




Por último la función OR puede crearse combinando las salidas de XOR y AND a través de una puerta OR. Lo demuestra la tabla de verdad que ofrecemos a continuación:

| A | B | Salida | Comentario |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 1 | 1 | Función XOR |
| 1 | 0 | 1 | |
| 1 | 1 | 1 | Función AND |

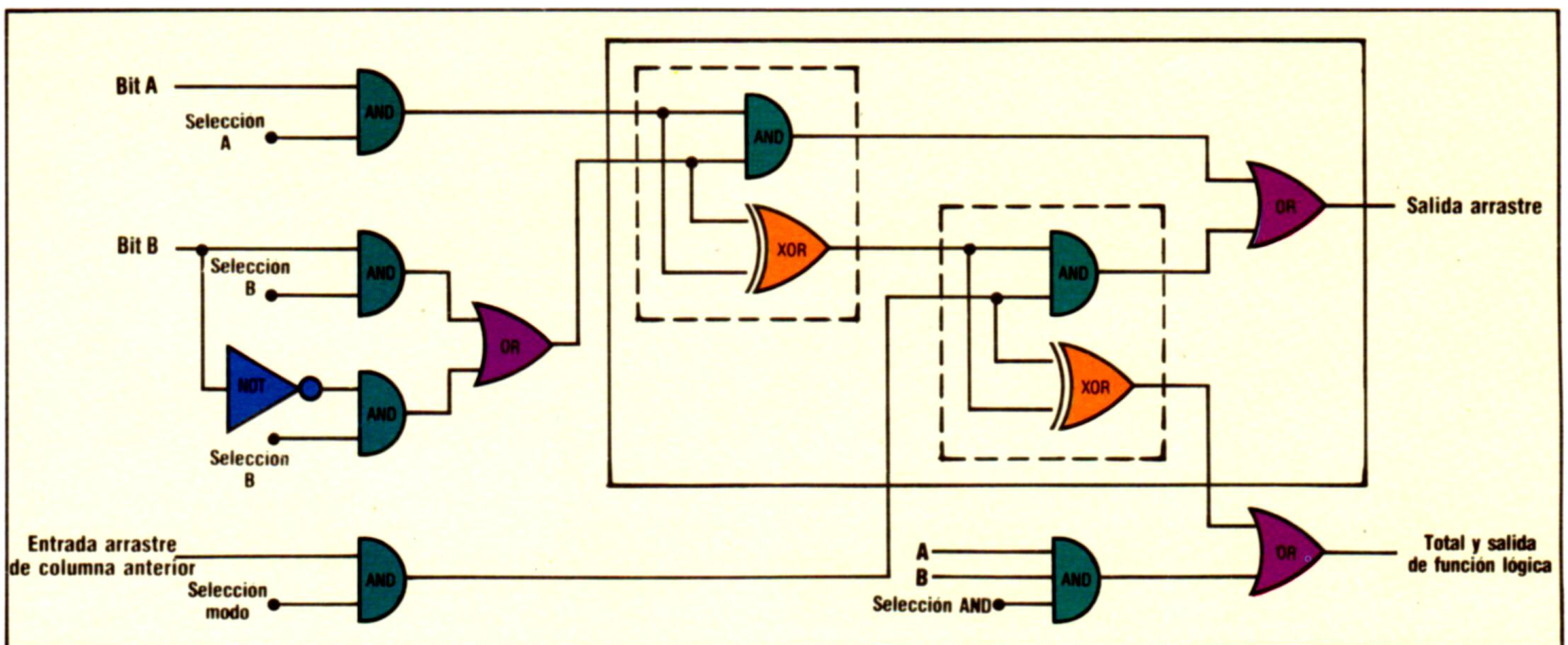
Esta otra tabla muestra la manera de obtener cada una de las funciones lógicas mediante combinaciones de las señales de control:

| Funcion | Selección modo | Selección A | Selección B | Selección $\bar{B}$ | Selección AND |
|---|---|---|---|---|---|
| XOR | 0 | 1 | 1 | 0 | 0 |
| AND | 0 | 0 | 0 | 0 | 1 |
| OR | 0 | 1 | 1 | 0 | 1 |

El diseño final que concluye este capítulo nos muestra un circuito de la ALU de un bit, que incluye el circuito sumador completo y demás circuitos adicionales para las señales de control. El circuito completo se compone de ocho circuitos como éste en paralelo. La salida de arrastre procedente de la octava columna sirve de indicador de arrastre al registro de estado del procesador (PSR).

Así concluimos nuestra serie de lecciones sobre lógica. Recuerde que la iniciamos con conceptos tan abstractos como el álgebra de Boole y los diagramas de Venn, para aplicarlos a circuitos lógicos muy sencillos, examinando sus resultados. Pero los últimos capítulos se alejaban de los conceptos abstractos y se referían ya a temas relaciondos con circuitos mucho más complicados a un nivel cercano al que realmente existe en un ordenador.

Para el final hemos dejado la explicación de cómo es posible combinar varios circuitos de tal modo que proporcionen al microprocesador los medios para realizar sus típicas operaciones aritméticas y lógicas. Operaciones que sólo exigen, para su perfecta ejecución, los patrones adecuados de unos y ceros en las líneas de instrucciones de la ALU. Estos patrones no son más que las instrucciones del lenguaje máquina en su auténtica forma binaria. Hemos, pues, estudiado la lógica del sistema físico de los ordenadores hasta el mismo borde donde empieza el terreno del software.




Diagramas: Liz Dixon

# El método adecuado

## En este último capítulo examinaremos la manera de utilizar, en micros basados en cassette, las técnicas descritas anteriormente

Cada micro utiliza su propia técnica de tratamiento de archivos, y, por lo tanto, a menudo es necesario adaptar un programa estándar para ejecutarlo en una máquina determinada. Por este motivo es importante saber qué relación hay entre las facilidades e instrucciones que ofrece su sistema con los métodos generales que hemos analizado. A modo de ejemplo, examinemos el almacenamiento de archivos de datos en un micro estándar basado en cassette. El primer punto a considerar es que los sistemas de cassette no pueden manipular archivos de acceso directo (véase p. 724), y hay que acceder a los datos por el orden en que están almacenados, es decir, secuencialmente.

Dado que el tratamiento de un archivo secuencial implica leer la información del archivo, trabajar sobre esta información y luego escribir los datos modificados en un segundo archivo, es obvio que debe haber dos archivos en uso ("abiertos") al mismo tiempo. Una grabadora de cassette es incapaz de moverse directamente y con precisión en dos porciones de cinta a la vez, de modo que este sistema de "dos archivos" no es apto para la mayoría de los micros basados en cassette. Las excepciones a esta regla son esos pocos micros, como el Newbrain y el Commodore PET, con dos puertas para cassette: una para leer, otra para escribir.

La mayoría de las máquinas personales están, por consiguiente, limitadas a un archivo secuencial por vez. En la práctica ello supone algunas limitaciones importantes. El archivo debe ser leído sobre la memoria antes de utilizarlo y luego, si se producen cambios, debe ser grabado de nuevo en cassette. Esto se podría hacer a intervalos durante la ejecución del programa, o una vez al finalizar su ejecución. Los archivos de datos deben ser suficientemente pequeños para residir en RAM en el espacio libre dejado por el programa que los trata. La mayoría de los micros personales están restringidos, por tanto, a archivos de datos de pequeño tamaño.

Se han desarrollado tres métodos principales para almacenar información en cassette. El sistema más simple no utiliza archivos de datos separados del programa, sino que todas las variables en curso se almacena junto con el programa cada vez que se utiliza la instrucción SAVE. Este procedimiento se emplea en el ZX81 y también está disponible para el Sinclair Spectrum. Cuando se requiere un nuevo archivo de datos se utiliza una copia "fresca" del programa, que después se guarda (SAVE) junto con sus datos particulares. Cuando esta versión se carga (LOAD) posteriormente, las variables quedan automáticamente con los valores que tenían en el momento del SAVE. La virtud de este método es su simplicidad: todo lo que el usuario tiene que hacer es asegurarse de que todo el programa se guarde (SAVE) y se cargue (LOAD) correctamente.

Un sistema ligeramente más sofisticado exige un BASIC que sea capaz de almacenar y volver a leer matrices específicas. En el Oric Atmos, por ejemplo, la orden STORE A$,NOMBRE" grabará la matriz A$ en cinta, y RECALL A$"NOMBRE" la volverá a leer sobre la memoria. Toda la matriz (A$(1), A$(2), etc.) se guarda (SAVE), a pesar de que las instrucciones STORE y RECALL no especifican las dimensiones de la matriz, que se determina cuando la matriz se DIMensiona al comienzo del programa.

Este sistema tiene el problema de que es necesario llevar la cuenta del número de entradas en la matriz que se utiliza en el programa. Una solución consiste en almacenar el contador de elementos usados antes de guardarla (SAVE). La mayoría de las máquinas admiten un subíndice cero, de modo que se puede emplear para el contador de registros el elemento A$(0,0). El contador de registros será una variable numérica (en nuestro programa utilizamos R), pero si se está utilizando una matriz de strings, ésta se debe convertir en un string. Esto es bastante fácil de hacer con una línea como: A$(0,0) = STR$(R). Después de que se haya vuelto a cargar la matriz en el ordenador, R se restablece mediante: R = VAL(A$(0,0).

Aunque muchos micros no admiten procedimientos sofisticados para tratamiento de archivos, éstos se pueden simular, como muestran los listados de la página siguiente. Una vez el archivo instalado en la memoria, es fácil usar matrices de BASIC para tratarlo como a un archivo de acceso directo.

Supongamos que se emplea una matriz de caracteres bidimensionales para almacenar los datos; ésta se podría definir con una instrucción como DIM A$(100,3). El primer subíndice de la matriz se podría utilizar para referenciar un registro determinado, y el segundo subíndice apuntaría al campo determinado dentro de ese registro. Esto permite almacenar los datos en el formato familiar de una tabla y equivale, en cuanto a operación, a un archivo de acceso directo.

Otra facilidad útil de que disponen algunas máquinas es la instrucción APPEND. Ésta permite que uno agregue datos al final el archivo secuencial sin tener primero que leerlo y crear luego una nueva versión. Algunos ordenadores disponen de una instrucción que permite saltarse un cierto número de campos en un archivo secuencial, lo que proporciona una facilidd de acceso directo simple.

Esta serie de capítulos ha hecho hincapié en los aspectos fundamentales de un tema complejo. El tratamiento de archivos depende mucho del ordenador y será necesario adaptar a su propia máquina todas las ideas que hemos ofrecido en estos capítulos. Pero los principios básicos serán los mismos, independientemente de qué micro se emplee, y cuando escriba sus propios programas gran parte del material le resultará muy útil.



# Almacenamiento de registros y campos en una matriz de BASIC

Una matriz de caracteres bidimensional se puede utilizar para emular un archivo de acceso directo. El primer subíndice se emplea para identificar un registro determinado y el segundo especifica campos dentro del registro

**Número de campos**
Es cómodo tener este número definido en una variable al principio del programa. Esto hace que resulte más fácil utilizar más tarde registros mayores, porque será necesario alterar una sola instrucción

**Registro de cabecera**
A$(0,0) se utiliza para almacenar un contador del número de registros. Cuando la matriz se guarda (SAVE) en cinta, este valor se almacena con ella

**(número de campo)**
0 1 2 3

**(número de registro)**
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 98 99 100

**Número de registros**
Una variable guarda el número de registros que están siendo usados en cada momento

**Número máximo de registros**
Se utilizará una variable para guardar el número máximo de registros que se pueden almacenar en la matriz. Éste es el número especificado en la sentencia DIM que crea la matriz

**Eliminación de un registro de la matriz**
Este fragmento del programa elimina de la matriz el registro número N utilizando el método que ya hemos detallado. Todos los registros por debajo de N se desplazan un lugar hacia arriba, machacando los datos almacenados en la posición N

```
200  FOR I=N TO R-1
210  FOR J=0 TO F
220  LET A$(I,J)=A$(I+1,J)
230  NEXT J:NEXT I
240  LET R=R-1
250  RETURN
```

**Inserción de un registro en la matriz**
Este otro fragmento inserta un nuevo registro en la matriz en la posición N del archivo. Todos los registros después del punto de inserción se desplazan hacia abajo para crear un vacío para el nuevo registro almacenado en N$, C$, D$ y E$

```
100  LET R=R+1
110  IF R>M THEN PRINT "MATRIZ LLENA":RETURN
120  FOR I=R TO N+1 STEP-1
130  FOR J=0 TO F
140  LET A$(I,J)=A$(I-1,J)
150  NEXT J:NEXT I
170  LET A$(N,0)=N$:LET A$(N,1)=C$
180  LET A$(N,2)=D$:LET A$(N,3)=E$
190  RETURN
```

Kevin Jones

# Cómo cargar y guardar una matriz de registros

**Variables guardadas con el programa**
El Spectrum guarda todas sus variables junto con los programas (aunque también puede guardar matrices solas). El programa de muestra llena una matriz con datos y la instrucción SAVE almacenará la matriz en cinta junto con el programa. Cuando éste se vuelve a cargar, empieza automáticamente por la línea 700, porque la instrucción SAVE lo remite a aquella línea. La instrucción RUN no se debe utilizar porque borra todas las variables

**Spectrum**

```
100   REM ***DEMO SPECTRUM***
200   DIM A$(100,20)
300   FOR K=1 TO 100
400   LET A$(K)="Numreg"+STR$(K)
500   NEXT K
600   STOP
700   PRINT"LA MATRIZ CONTIENE  "
800   FOR K=1 TO 300
900   PRINT A$(K)
1000   NEXT K
110   STOP
SAVE "PROGDEMO" LINE 700
LOAD "PROGDEMO"
```

**Cómo guardar matrices con nombre**
El Oric Atmos dispone de las instrucciones STORE y RECALL para guardar y cargar en cinta matrices concretas. Gracias a estas instrucciones es fácil almacenar un archivo contenido en una matriz

**Oric Atmos**

```
100  REM Guardar matriz en cinta
110  A$(0,0)=STR$(R)
120  PRINT "Por favor posicione la cinta, pulse
     PLAY y RECORD y después RETURN"
130  A$=KEY$:IF A$="" THEN 130
140  STORE A$,"ARCHIVO",S
150  PRINT "CONCLUIDO"
160  RETURN

200  REM Cargar matriz desde cinta
210  PRINT "Por favor posicione la cinta y pulse
     PLAY, luego RETURN"
220  A$=KEY$:IF A$="" THEN 220
230  RECALL A$,"ARCHIVO",S
240  R=VAL (A$(0,0))
250  PRINT "CONCLUIDO"
260  RETURN
```

**Utilización de archivos seriales**
El BBC Micro es uno de los diversos ordenadores personales que admiten auténticos archivos secuenciales en cassette. Estas dos subrutinas almacenan y vuelven a cargar la matriz mediante la creación de un archivo secuencial. El primer campo es el contador de registros

**BBC Micro**

```
100  REM Guardar matriz en cinta
105  PRINT "Por favor posicione la cinta"
110  X=OPENOUT ("ARCHIVO")
120  PRINT £X,R
130  FOR I=1 TO R
140  FOR J=1 TO F
150  PRINT £X,A$(I,J)
160  NEXT J:NEXT I
170  CLOSE £X
180  PRINT "CONCLUIDO"
190  RETURN

200  REM Cargar matriz desde cinta
205  PRINT "Por favor posicione la cinta y pulse
     PLAY después RETURN"
210  IF INKEY(0)=-1 THEN 210
215  X=OPENIN ("ARCHIVO")
220  INPUT £X,R
230  FOR I=1 TO R
240  FOR J=1 TO F
250  INPUT £X,A$(I,J)
260  NEXT J:NEXT I
270  CLOSE £X
280  PRINT "CONCLUIDO"
290  RETURN
```

# La pulga, o ¿cómo salir del abismo?

## "Bugaboo" (La pulga) es un insólito juego para el Spectrum y el Commodore 64: el jugador es una pulga que queda atrapada en un gran agujero

La impresionante secuencia de títulos del juego *Bugaboo*, de Quicksilva, le informa que se está aproximando al planeta Cebella-7, en el que existen indicios de vida. Después de aterrizar con toda seguridad, usted va brincando por la fantástica flora de la superficie del planeta hasta perder pie y despeñarse en un profundo agujero, que se abre a una enorme caverna. El objetivo del juego es escapar del abismo.

Unos salientes que sobresalen de rocas de llamativos colores permiten apoyar los pies en un paisaje que recuerda las pinturas del Bosco: setas y flores multicolores proliferan por doquier y las arañas trepan por las paredes de la caverna. Usted ha de intentar escapar saltando de saliente en saliente, evitando a la vez a un dragón devorador de pulgas. Sólo en la versión para el Commodore, las plantas atrapamoscas proporcionan un nuevo aliciente: otro obstáculo floral.

### A saltar

La característica más sobresaliente del *Bugaboo* son sus detallados gráficos. Éstos representan un profundo agujero que tiene muchos salientes entre los cuales el jugador (que personifica a una pulga) tiene que ir saltando para poder ascender hacia la libertad. Los gráficos del Commodore le llevan una ligera ventaja a los de la versión para el Spectrum




Bugaboo (La pulga) en el Spectrum



Bugaboo (La pulga) en el Spectrum




Booga-boo (La pulga) en el Commodore

En ambas versiones, la pantalla actúa como una ventana de la zona de juego, y al jugador se le van presentando nuevas secciones de la caverna a medida que se va desplazando y va trepando hacia arriba. La caverna tiene un ancho aproximado de tres pantallas y una altura de alrededor de cinco pantallas, de modo que descubrir todos los obstáculos lleva su tiempo. Se puede tomar una cualquiera de varias frutas diferentes y los salientes están situados de forma muy astuta para aumentar el grado de dificultad.

La escenografía es original, pero el juego pierde mucho debido a la codificación del programa.

El jugador sólo tiene una vida, y la secuencia de instrucciones de la introducción se repite cada vez que uno pierde la vida, lo que sucede con muchísima frecuencia, ya que evitar al dragón resulta casi imposible.

A medida que avanza el juego, los salientes son cada vez más difíciles de alcanzar. Pero en los niveles superiores no se introducen obstáculos, lo que es lamentable, ya que el programa produciría mucha más adicción si en las sucesivas etapas fueran apareciendo otros depredadores.

El almacenamiento de las múltiples pantallas utiliza muchísima memoria, dejando poca para el uso de reglas complicadas y acción, pero hay otros muchos programas que consiguen más: *Jet Set Willy* es un buen ejemplo. La versión para el Commodore 64 realiza gráficos más complejos que la versión para el Spectrum, pero, por el contrario, no consigue obtener el máximo partido de su memoria adicional.

La versión para el Spectrum se controla mediante el teclado, haciendo que las teclas "1" y "0" controlen los saltos a izquierda y derecha. La fuerza de estos saltos está determianda por el tiempo durante el cual se mantenga pulsada la tecla en cuestión: cuánto más tiempo se mantenga pulsada la tecla, más arriba se saltará.

La versión para el Commodore 64 es para utilizar sólo con una palanca de mando, con el pulsador de disparo controlando la exploración: un arreglo más satisfactorio en caso de que se posea una palanca de mano.

A pesar de las críticas hechas al *Bugaboo* acerca de su calidad de adicción a largo plazo, hay que decir que el juego es muy recomendable. Los controles son sumamente sencillos, permitiendo que el usuario empiece a jugar de inmediato, sin tener que remitirse una y otra vez a la pantalla de instrucciones. Y los imaginativos gráficos son, sencillamente, magníficos.

# Artimañas aritméticas

**Esta vez estudiaremos cómo resta y multiplica un microprocesador. Para ello debemos presentar dos operaciones lógicas: desplazamiento y rotación**

Decíamos en el capítulo anterior que tanto el 6502 como el Z80 disponen de la instrucción SBC (*SuBtract with Carry*: restar con arrastre), pero el modo de ejecutarla es distinto. En el 6502 el flag de arrastre sirve para "quitar uno", de la misma forma que servía para "añadir uno" en la suma. En el assembly del Z80, sin embargo, el funcionamiento de SBC se corresponde perfectamente con el de ADC: el flag se pone a 1 o queda a cero, según sea el resultado de la operación.

Así, por ejemplo, si usamos ADC para obtener la suma $E4+$5F (poniendo antes, como de costumbre, el flag a cero) en el acumulador obtendremos $43 y el flag de arrastre estará a 1, lo cual nos está indicando que el resultado completo es $0143. Ha habido un desbordamiento (*overflow*) que activó al flag ya que el resultado no cabía en el acumulador, capaz tan sólo de un byte.

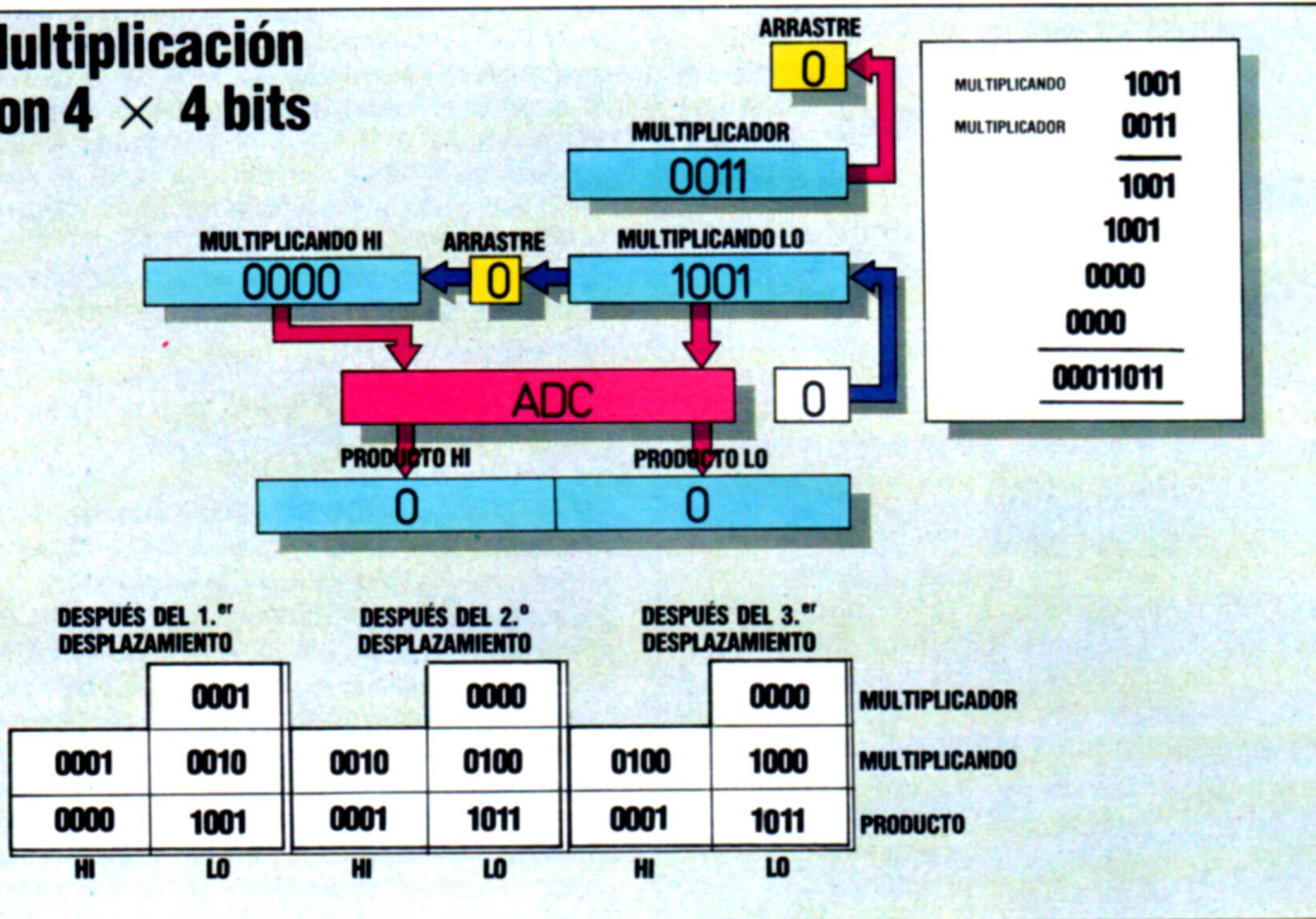
Supongamos que deseamos realizar con el Z80 la resta $5F−$E4. En el acumulador obtendremos $7B y el flag se activará. ¿Qué resultado es éste? En decimal equivale a escribir 95 − 228 + 123 (valor del hexa 7B), pero el significado que para el Z80 tiene el flag activado después de la resta nos permite dar con el resultado verdadero. El flag en 1 significa que debemos tomar el número contenido en el acumulador como *el complemento a 2* del resultado verdadero, ya que nos está diciendo que se ha producido un resultado negativo. Para hallar la solución hemos de "desandar" el camino que en el capítulo anterior describimos hasta dar con el complemento a 2. Veámoslo:

| | |
|---|---|
| $7B en binario | = 01111011 |
| quitamos 1 | − 1 |
| complemento a 1 | 01111010 |
| cambio de unos y ceros | |
| y obtenemos el compl. a 2 | 10000101 = $85 |

Ya tenemos el resultado auténtico negativo: el número −$85 (decimal, −133).

Naturalmente, las cosas son distintas cuando la resta se realiza a dos bytes. Habremos de distinguir entre los bytes inferiores (los llamados bytes *lo* y los bytes superiores (bytes *hi*). Por ejemplo:

| HI LO | | |
|---|---|---|
| $37 5F | = | 14175 en decimal |
| − $21 E4 | = | −8676 en decimal |
| $15.7B | = | 5499 en decimal |




**Desplazamientos**
Este ejemplo muestra, para mayor claridad, una multiplicación con sólo 4 bits para cada número. El algoritmo no cambia si hay más bits. Observe cómo se forma el producto por medio de la suma de ceros o del multiplicando desplazado, según el bit del multiplicador sea cero o uno. Los bits del multiplicador se desplazan hacia la derecha pasando por el flag de arrastre, mientras que los bits del multiplicando se desplazan hacia la izquierda pasando del byte *lo* al byte *hi* a través del flag de arrastre

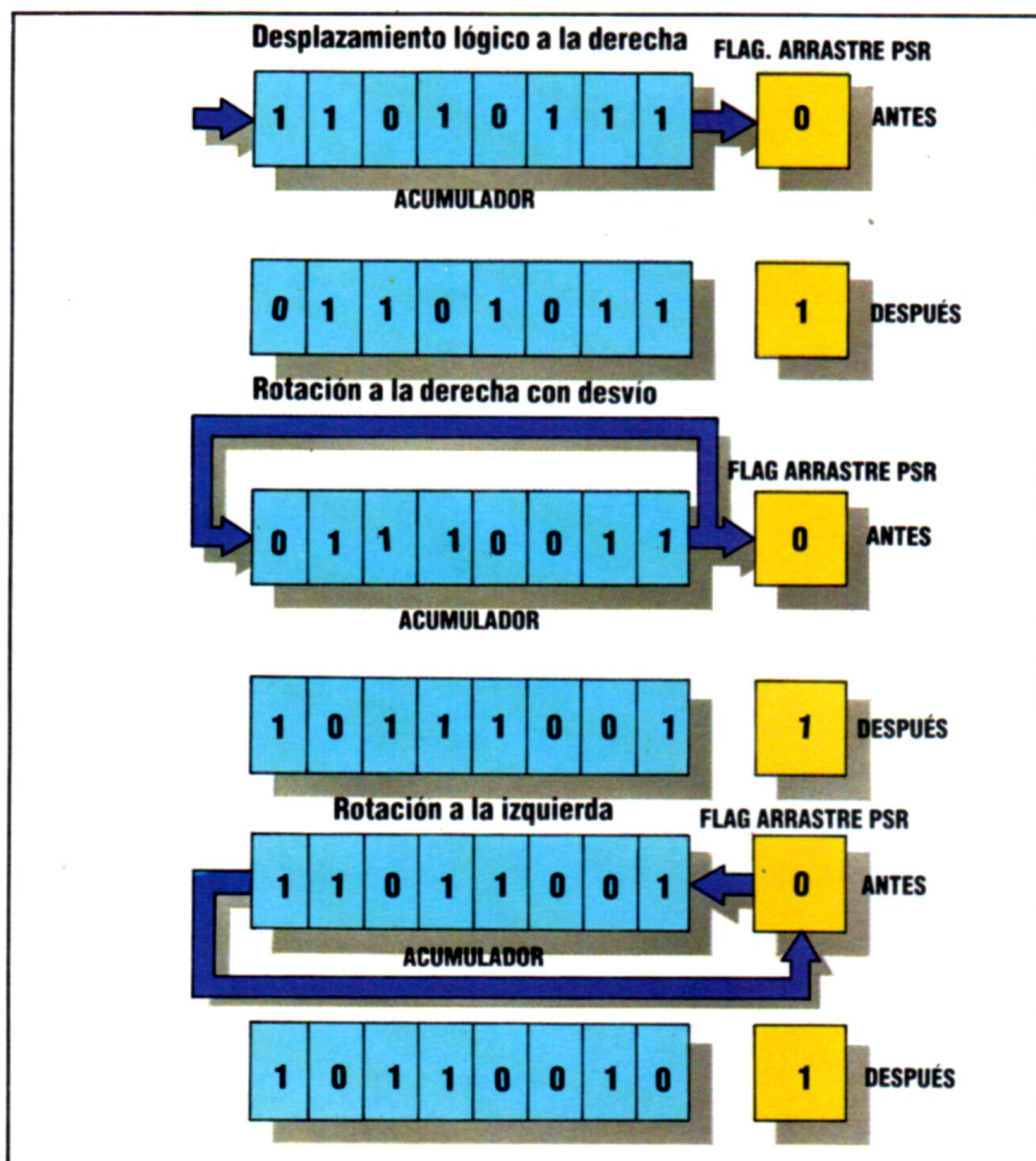
En restas de este tipo, la resta en los bytes *lo* sabemos que da $7B en el acumulador y la activación del flag de arrastre. Esto hace que en los bytes *hi* el sustraendo $21 se convierta en $22, que restado con el minuendo $37 da $15. Vemos que el resultado $375F−$21E4=$157B es correcto, según comprobamos por su versión decimal.

Resumiendo, la operación a dos bytes en un Z80 sigue un procedimiento muy sencillo, que ofrecemos a continuación:

1) Se pone a 0 el flag de arrastre.
2) Se restan los bytes *lo* con arrasstre.
3) Se restan los bytes *hi* con arrastre.

El 6502 se diferencia sobre todo en el paso 1). Lo que se hace previo a la resta es poner el flag en 1, pues en 0 significa que se debe quitar una unidad en los bytes *hi*, según resulta de la resta de los bytes *lo*. Si éstos no necesitan tomar ninguna unidad de aquéllos, entonces la resta transcurre sin más dificultades, y el flag permanece en 1 preparado para la resta de los bytes *hi* que deberá realizarse con idéntica normalidad. Pero si en la resta de los bytes *lo* se tuviera necesidad de una unidad adicional, el flag de arrastre actuaría como un "noveno bit" del acumulador. Lo cual permite que se obtenga el resultado correcto y pone a 0 el flag. En este estado el efecto sobre la resta de los bytes *hi* es idéntico al que realiza el flag activado en el Z80: el número minuendo sufre una disminución de una unidad antes de ser restado. Ambos métodos recuerdan el tradicional "no cabe, me llevo diez, resto y quito una" de nuestras restas infantiles. Veamos la versión para el 6502 más detenidamente.

**Instrucciones**
Las instrucciones de desplazamiento y rotación sirven ante todo para examinar el contenido de un registro bit a bit. A cada desplazamiento, el bit del extremo (izquierda o derecha) se traslada al flag de arrastre del registro indicador de estado (PSR), pudiéndose emplear su estado para ejecutar una bifurcación del flujo del programa. Las instrucciones de rotación se emplean cuando queremos conservar el contenido del registro, pero las instrucciones de desplazamiento lógico van colocando ceros por un lado a medida que desplazan los bits hacia afuera por el otro. Por tanto, un desplazamiento a la izquierda multiplica por dos el contenido del registro, y un desplazamiento a la derecha lo divide por dos




Si mantuviéramos el flag de arrastre en 0 y realizáramos la resta $5F−$E4, en el acumulador tendríamos, para nuestra sorpresa, $7A y el flag no se inmutaría. Pero ya hemos visto que el "supuesto" resultado verdadero es $7B más una señal del flag de que el número es negativo (o sea, que es el complemento a 2 del resultado definitivo). Pero $7A es el complemento a 1, si bien se ve, de dicho resultado definitivo. De lo que se deduce que el estado del flag indica el complemento a 2 si está activado, y el complemento a 1 si está a cero.

Si, por el contrario, activamos antes de nada el flag y después hacemos la resta, el acumulador contendrá $7B y el flag se pondrá en cero. Para el caso de que la resta sea de dos bytes y no se pare aquí, el flag en cero se encarga de quitar una unidad al byte *hi* minuendo: la unidad que se tomó de más en la resta de los bytes *lo*.

## Multiplicación

**Examinemos una multiplicación en decimal:**

| | |
|---|---|
| 174 | multiplicando |
| × 209 | multiplicador |
| 1566 | 1.^er^ producto parcial |
| 000 | 2.º producto parcial |
| + 348 | 3.^er^ producto parcial |
| 36366 | producto total |

No hace falta, para entender bien este método, más que seguirlo en su modo de disponer los productos parciales y saberse la tabla de multiplicar. La multiplicación se reduce a una suma si se ha cumplido su norma más fundamental: la de ir colocando cada producto parcial desplazado un lugar hacia la izquierda respecto de la situación del anterior producto parcial (incluimos, para mayor claridad, el producto con sólo ceros).

Esta alternancia de productos parciales desplazados y tablas de multiplicar es lo que nos hacía difícil la multiplicación en nuestros años de escuela. Pero en binario la tabla de multiplicar es sencillísima: $1 \times 1 = 1$ y $1 \times 0 = 0$. Sólo nos queda respetar la colocación

| | |
|---|---|
| 1101 | (decimal, 13) |
| × 1001 | (decimal, 9) |
| 1101 | 1.^er^ prod. parcial |
| 0000 | 2.º prod. parcial |
| 0000 | 3.^er^ prod. parcial |
| 1101 | 4.º prod. parcial |
| 1110101 | (decimal, 117 = 13 ×9) |

Se observan claramente en este ejemplo los desplazamientos de los productos parciales, así como la extremada sencillez de la multiplicación en binario. Note que un producto parcial cualquiera o es todo ceros o es idéntico al multiplicando. Lo que inmediatamente nos recuerda el test al que estamos acostumbrados como prueba del lenguaje assembly. Para realizar una multiplicación binaria, debemos examinar cada uno de los bits del multiplicador por orden, e ir sumando ceros (si el bit es cero) o el multiplicando desplazado (si el bit es uno) al total.

Vamos a ver cómo se pueden examinar los bits, uno a uno, del multiplicador y cómo se desplaza el multiplicando.

La comprobación del estado de un bit en particular dentro de un byte se puede realizar, en ambos microprocesadores, el Z80 y el 6502, mediante la instrucción BIT. En el Z80 esta instrucción tiene como operandos una dirección y el número del bit a investigar, poniendo el flag de cero a 0 s el bit es uno, y a 1 si el bit es cero. En el 6502 el operando es sólo una dirección. Con el contenido de esta dirección realiza la operación lógica AND sirviéndose del acumulador, y según el resultado sea verdadero o falso del flag de cero permanece en cero o cambia a uno.

Estas instrucciones permitirían una acertada programación, pero ningún método nos conviene por el momento. Mejor sería si el bit de que se trata nos sirviera como flag de arrastre o flag de cero, de tal modo que el flujo del programa se desviara automáticamente según el estado de cada uno de los bits. Esto naturalmente es posible con las instrucciones de que disponen ambos procesadores, y en particular con las "instrucciones de desplazamiento" (shift instructions). Según ya indica su nombre, permitirán también resolver el problema de cómo desplazar el multiplicando.

Varias son las instrucciones de "desplazamiento" (*shift*) o de "rotación (*rotate*) en ambos procesadores. Tienen por efecto general el de desplazar cada bit contenido en un registro un lugar hacia la izquierda o hacia la derecha. Las diferencias estriban en el modo de tratar los bits que están en los extremos del registro: un bit debe desplazarse fuera del registro por un extremo mientras que por el opuesto hay que emplazar otro bit. En el caso de que se desplace fuera el bit 7 y este mismo aparezca inmediatamente como bit 0, tendríamos una "rotación a la izquierda" (*rotate left*). Lo contrario, el bit 0 va desplazando al bit 7, será una "rotación a la derecha" (*rotate right*). Es claro que tras ocho rotaciones del mismo tipo volvemos a obtener el registro de partida.

Si no se utiliza la rotación debemos encontrar un destino para el bit desplazado fuera del registro, y una fuente de donde sacar el bit de relleno. La mayoría de las veces, ambas cosas las suministran los diversos flags, condición del registro indicador de estado (PSR), y en concreto el flag de arrastre. Para construir una subrutina que multiplique números de dos bytes es necesario desplazar el multiplicando a la izquierda y el multiplicador a la derecha. Los bits del multiplicando irán siendo desplazados hacia el byte *hi* de éste, mientras van apareciendo ceros en los bits que se vacían. Por su parte, los bits del multiplicador irán pasando por un flag del PSR para su comprobación, pero su destino final así como el estado de los bits que van rellenando el multiplicador según se va vaciando no tiene mayor interés, a menos que no necesitemos conservar el multiplicador en su forma primitiva. Lo que sí es relevante para lo que nosotros queremos es saber si el bit que acaba de desplazarse fuera del multiplicador es un uno o un cero.

Indicando que el multiplicador se encuentra en la dirección MPR, el multiplicando en MPDLO y el producto en PRODLO y PRODHI, podemos describir estas subrutinas del modo como expresamos a continuación:

**MULTIPLICACIÓN CON OCHO BITS**

| 6502 | | | Z80 | | |
|---|---|---|---|---|---|
| | ORG | $C100 | | ORG | $D000 |
| START | LDA | #$00 | START | LD | BC,(MPR) |
| | STA | PRODLO | | LD | B,$08 |
| | STA | PRODHI | | LD | DE,(MPDLO) |
| | STA | MPDHI | | LD | D,$00 |
| | LDX | #8 | | LD | HL,$00 |
| | CLC | | LOOP0 | SRL | C |
| LOOP0 | ROR | MPR | | JR | NC,CONT0 |
| | BCC | CONT0 | | CALL | ADDIT |
| | JSR | ADDIT | CONT0 | SLA | E |
| CONT0 | ASL | MPDLO | ENDLP0 | DJNZ | LOOP0 |
| | ROL | MPDHI | | LD | PRODLO |
| | DEX | | | RTS | |
| ENDLP0 | BNE | LOOP0 | MPR | DB | $E2 |
| | RTS | | MPDLO | DB | $7A |
| MPR | DB | $E2 | MPDHI | DB | $00 |
| MPDLO | DB | $7A | PRODLO | DW | $0000 |
| MPDHI | DB | $00 | ADDIT | ADD | HL,DE |
| PRODLO | DB | $00 | | RET | |
| PRODHI | DB | $00 | | | |
| ADDIT | CLC | | | | |
| | LDA | PRODLO | | | |
| | ADC | MPDLO | | | |
| | STA | PRODLO | | | |
| | LDA | PRODHI | | | |
| | ADC | MPDHI | | | |
| | STA | PRODHI | | | |
| | RTS | | | | |

Por el ejemplo podemos ver que la programación del Z80 es mucho más fácil gracias a sus registros de 16 bits y las instrucciones asociadas. Compare la subrutina ADDIT en ambos programas. La versión del 6502 emplea ROR (*rotate right*) para hacer rotar el multiplicador hacia la derecha pasando por el flag de arrastre, y se sirve de ASL (*shift left*) y ROL (*rotate left*) para desplazar el multiplicando hacia la izquierda dejando MPDLO y pasando a MPDHI a través del flag de arrastre. El bucle se controla con el registro X actuando de contador.

La versión del Z80 emplea SRL para desplazar el multiplicador hacia la derecha pasando por el flag de arrastre, y SLA y RL para desplazar el multiplicando contenido en DE hacia la izquierda pasando por el flag de arrastre. El control del bucle está a cargo del registro B que hace de contador. La instrucción ADD suma a dos bytes y no es afectada por el flag de arrastre (al contrario de ADC).

El próximo capítulo tratará de la división y de las diversas maneras de controlar la pantalla en las visualizaciones. Quedará así cubierta la teoría y podremos dedicarnos a realizar ejercicios con el 6502 y el Z80 en las siguientes lecciones.

## Ejercicio 15

**1)** Escriba una subrutina para multiplicar con un multiplicando de 16 bits y un multiplicador de 8 bits elegidos por usted.

**2)** La multiplicación no es sin la suma reiterada. Escriba una subrutina de multiplicación de 8 bits por 8 bits, sin recurrir a las instrucciones de desplazamiento y rotación.

# Elegancia italiana

**Olivetti, la prestigiosa multinacional europea, actualmente está considerada como una de las principales empresas especializadas en automatización de oficinas**



**Ángulo de diseño**
El ordenador de oficina Olivetti M20 es una máquina de 16 bits que salió al mercado en 1981. Incorpora un microprocesador Zilog Z8000 y utiliza el sistema operativo Olivetti PCOS. Recientemente Olivetti ha lanzado una versión de esta máquina que es compatible con el IBM-PC

Camillo Olivetti fundó su empresa en 1908. Con una plantilla de 20 empleados, instaló una tienda de máquinas en Ivrea, entonces una pequeña ciudad rural del norte de Italia, y comenzó a producir la primera máquina de escribir de la empresa, la M1. En aquellos tiempos la economía italiana, todavía era básicamente agrícola y carecía de la industria pesada que había favorecido la expansión en Alemania, Gran Bretaña y Estados Unidos. A pesar de esto, la producción de Olivetti fue aumentando de forma constante, pasando de lanzar al mercado cuatro máquinas diarias en 1914 a producir 50 por día en el año 1929.

En los años treinta Adriano Olivetti, hijo de Camillo, comenzó a reorganizar la empresa, introduciendo alumnos de la propia escuela nocturna de Olivetti, que se había fundado en 1924. También se realizaron mejoras sociales y se proporcionaría el alojamiento por parte de la empresa, una política "de la cuna a la tumba" que recuerda a la que luego utilizarían con tanta eficacia los japoneses. Mientras en el resto del mundo la industria luchaba inmersa en la depresión económica de entreguerras, Olivetti seguía creciendo, y hacia 1933 la empresa había vendido 15 millones de productos para oficina. En 1937 se lanzó la primera teleimpresora Olivetti, a la que siguió, en 1940, la primera calculadora de la empresa.

La segunda guerra mundial supuso una interrupción temporal de la expansión de Olivetti, pero en los años de la posguerra la empresa se concentró en el desarrollo de nuevos mercados. El éxito de la empresa se cimentaba en la elegancia de sus diseños y en la calidad de los productos, e incluso los ejecutivos de IBM se vieron obligados a admitir que los productos Olivetti "se acoplaban entre sí como un hermoso puzzle de imágenes".

En la década de los cincuenta y de los sesenta, Olivetti se concentró en el desarrollo de ordenadores de oficina. Este proceso empezó con la introducción de una máquina de calcular numérica, en 1955, y unos años después, producía el primer ordenador central, el Elea.

La empresa siguió diversificándose, abandonando la maquinaria mecánica de oficina y volcándose hacia los equipos basados en la electrónica, que Olivetti identificó con la principal tendencia de la automatización de oficinas. Se introdujo una nueva gama de microordenadores, a la cual no tardaría en agregarse la de los terminales bancarios y equipos de comunicaciones.

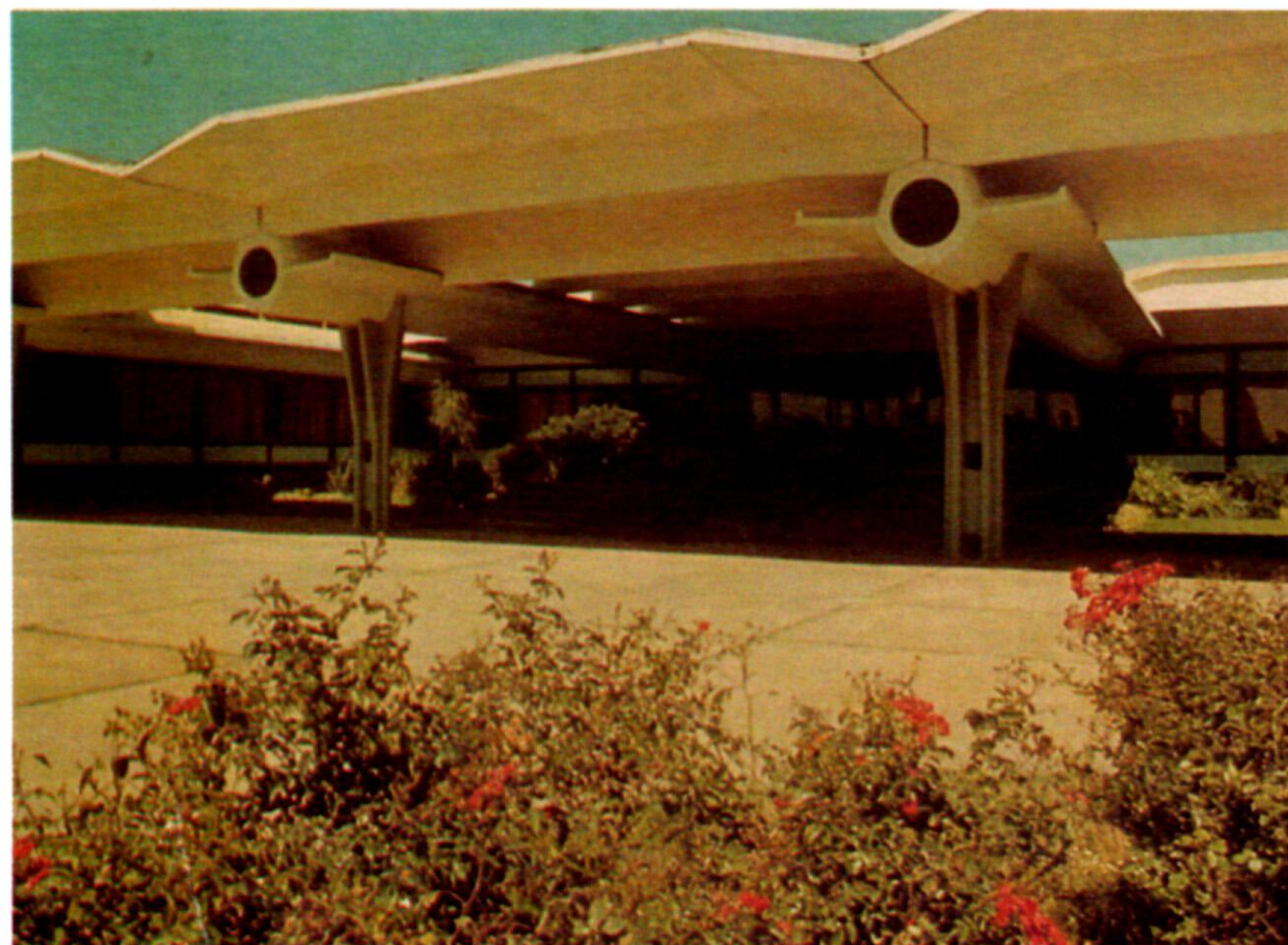
En la actualidad Olivetti fabrica una amplia gama de máquinas de oficina electrónicas, y la empresa invierte grandes sumas de dinero en desarrollar software de apoyo para sus máquinas. En 1982 Olivetti fue el segundo mayor fabricante de ordenadores de Europa (sólo superado por IBM), con el ordenador portátil M10 y la máquina de oficina M20. Tanto una como otra obtuvieron una excelente acogida en el mercado.

El ordenador de mano M10 pesa alrededor de 1,7 kg y viene equipado con una pantalla de 8x40 caracteres. La máquina funciona a pilas y tiene 8 Kbytes de RAM interna que se pueden ampliar a 64 Kbytes. La máquina de oficina M20, de 16 bits, trabaja sobre un microprocesador Z8001, no muy apreciado por otros fabricantes de máquinas de 16 bits. Asimismo, contiene un procesador 8086 que permite alguna compatibilidad con CP/M-86 y MS-DOS.

Olivetti también está planeando una nueva máquina compatible con el IBM-PC que, según la empresa, valdrá menos que su rival de IBM. Llamado M24, posee un procesador 8086-2 y tiene la opción de una tarjeta Z8001 para que sea compatible con el Olivetti M20. Esta compatibilidad ha tenido como consecuencia que Olivetti se viera en la obligación de abandonar su propio sistema operativo PCOS.

Recientemente Olivetti ha firmado un acuerdo con AT & T (la empresa de telecomunicaciones más grande del mundo) para colaborar en un proyecto para desarrolar el sistema operativo Unix. Y no cabe ninguna duda de que en el futuro la red de ventas internacional y el activo departamento de investigación y desarrollo de Olivetti mantendrán el prestigio de que goza la empresa a causa de la óptima factura, de la descollante calidad de todos sus productos para oficina.

**Exterior elegante**
Olivetti siempre se ha destacado por el avanzado diseño de sus productos. Pero su preocupación por el factor elegancia también alcanza a la arquitectura de los edificios de la empresa. Esta oficina se construyó en 1959 y muchas de sus características fueron posteriormente adoptadas por otros arquitectos




mi computer
Editorial Delta, S.A.


E R T Y U I O
D F G H J K L
X C V B N M


10039
9 788485 822836